STQA_Session_06

--Data flow testing

Dataflow Coverage

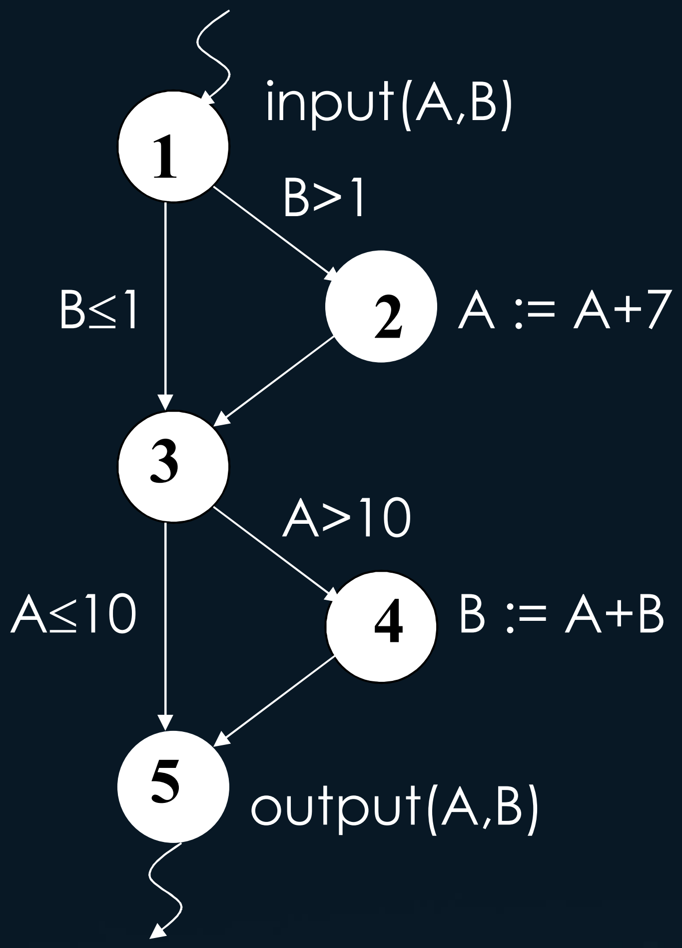
- Based on the idea that program paths along which variables are defined and then used should be covered.
- A family of path selection criteria has been defined, each providing a different degree of coverage.
- CASE tool support is very desirable.

Other Dataflow Terms and Definitions

- A path is *definition clear* **("def-clear")** with respect to a variable v if there is no **re-**definition of v within the path.
- A *definition-use pair* **("du-pair")** with respect to a variable v is a double (d,u) such that d is a node in the program's flow graph at which v is defined, u is a node or edge at which v is used, and there is a def-clear path with respect to v from d to u.
- (Note that the definition of a du-pair does not require the existence of a **feasible** def-clear path from d to u.)

Example 1

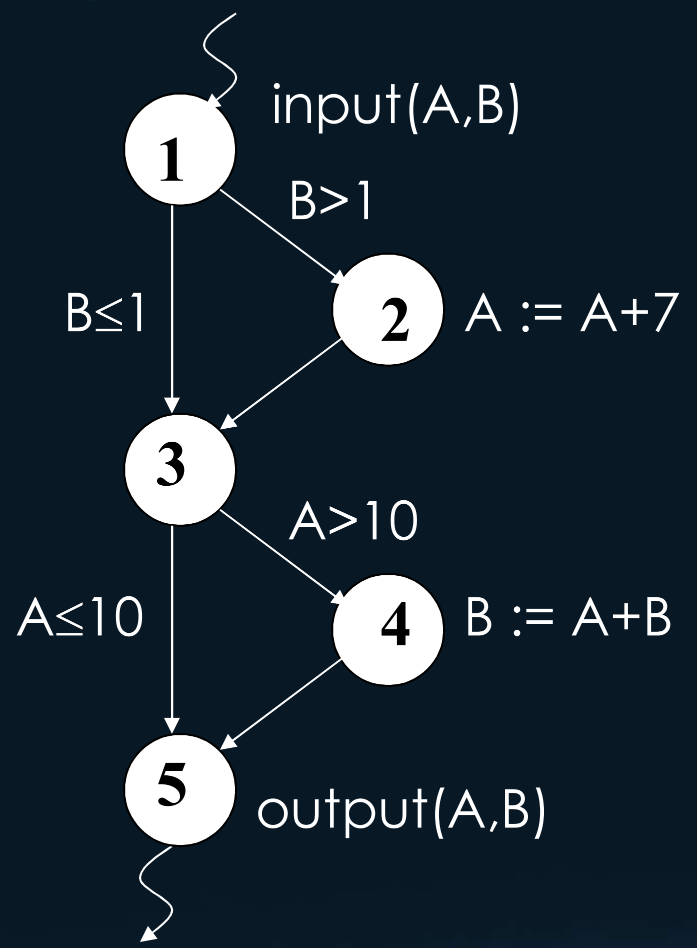
1. input(A,B)
   if (B>1) then
2.   A := A+7
   end_if
3. if (A>10) then
4.   B := A+B
   end_if
5. output(A,B)



Identifying DU-Pairs – Variable A

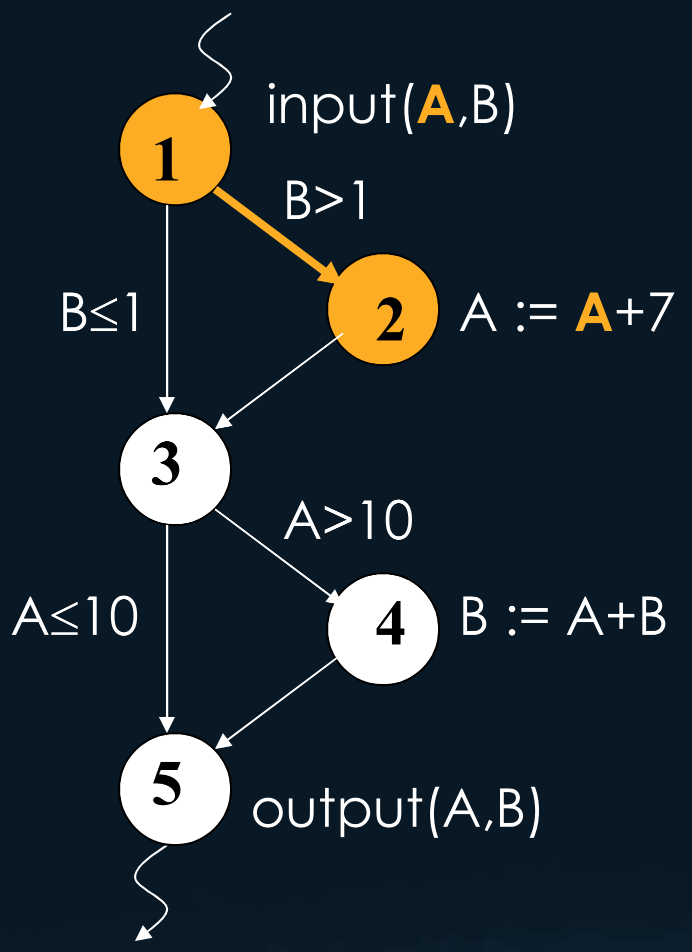
d-c paths

du-pair	**path(s)**
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



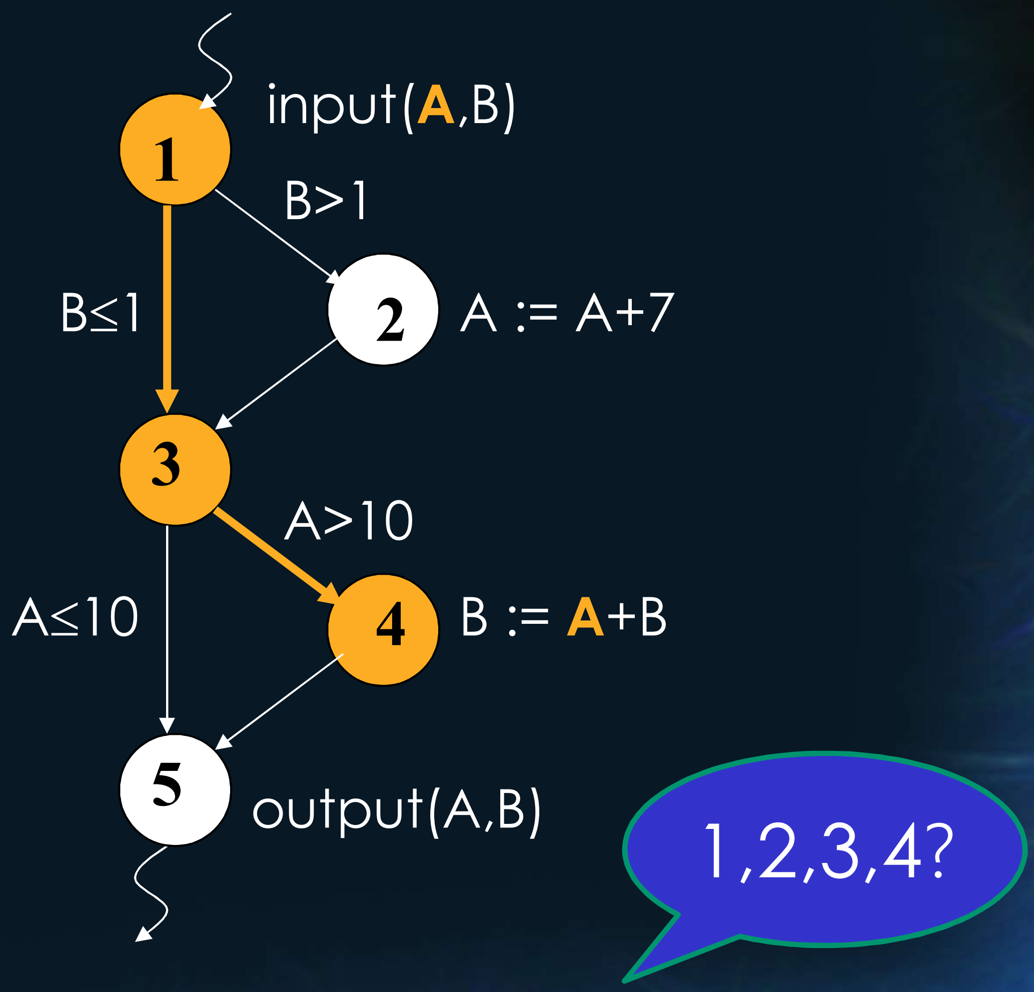
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



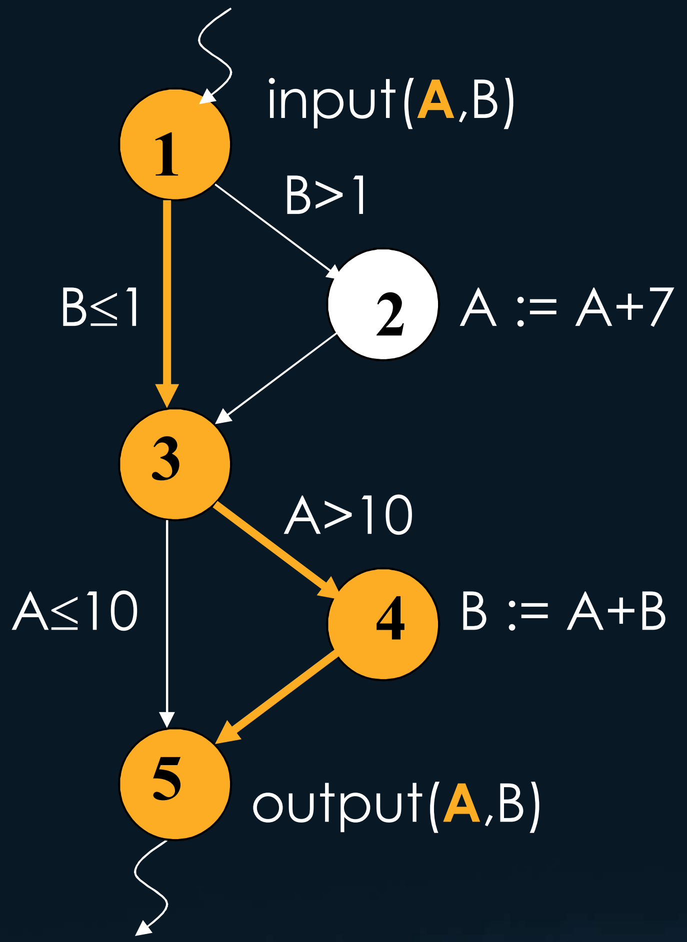
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	**<1,3,4>**
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



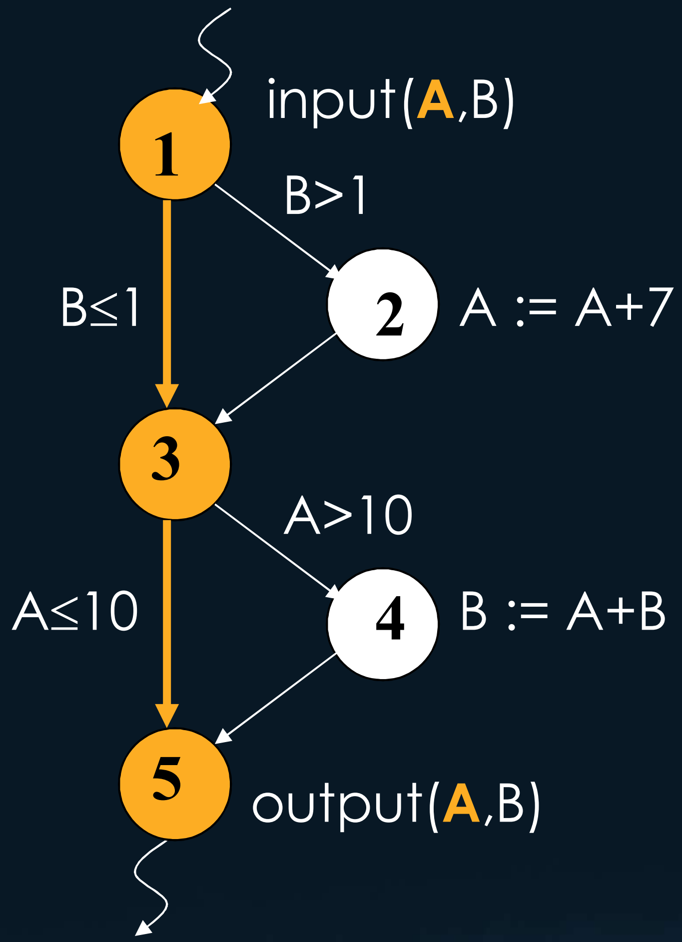
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	**<1,3,4,5>**
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



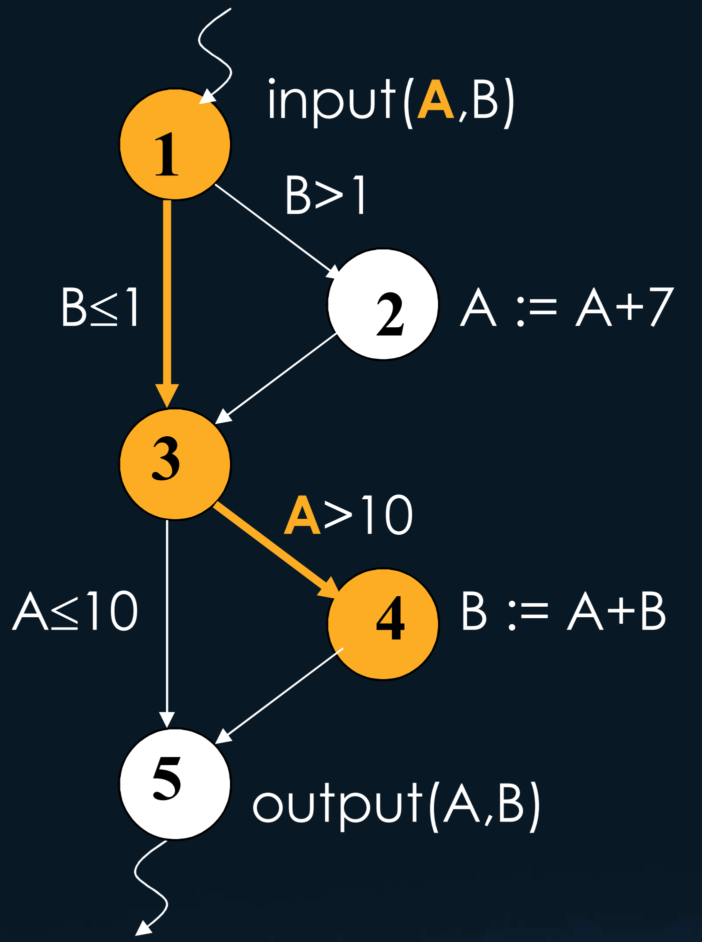
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



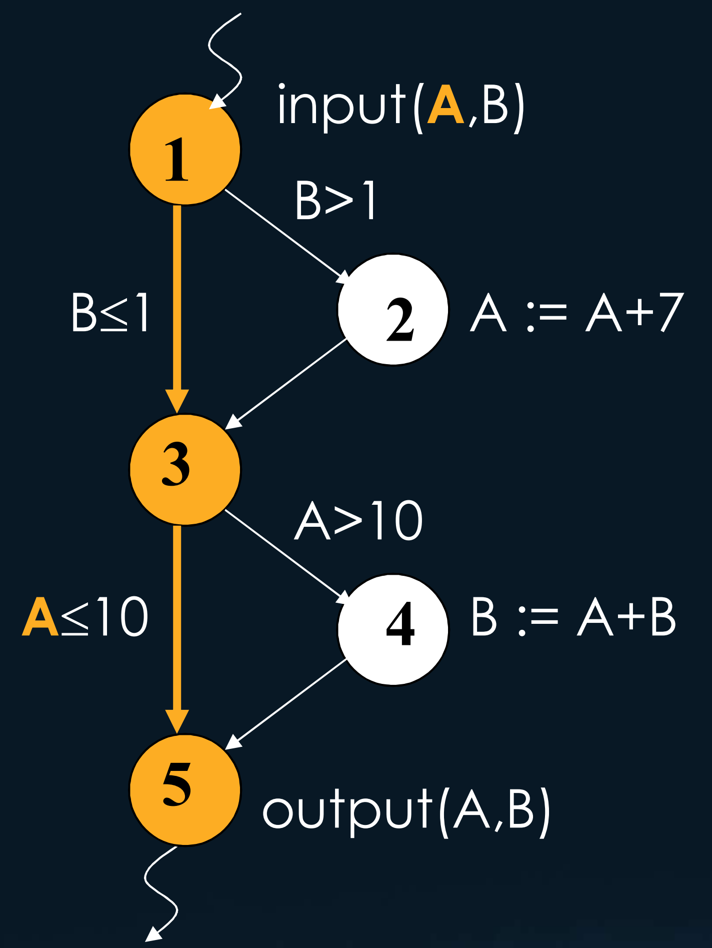
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	**<1,3,4>**
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



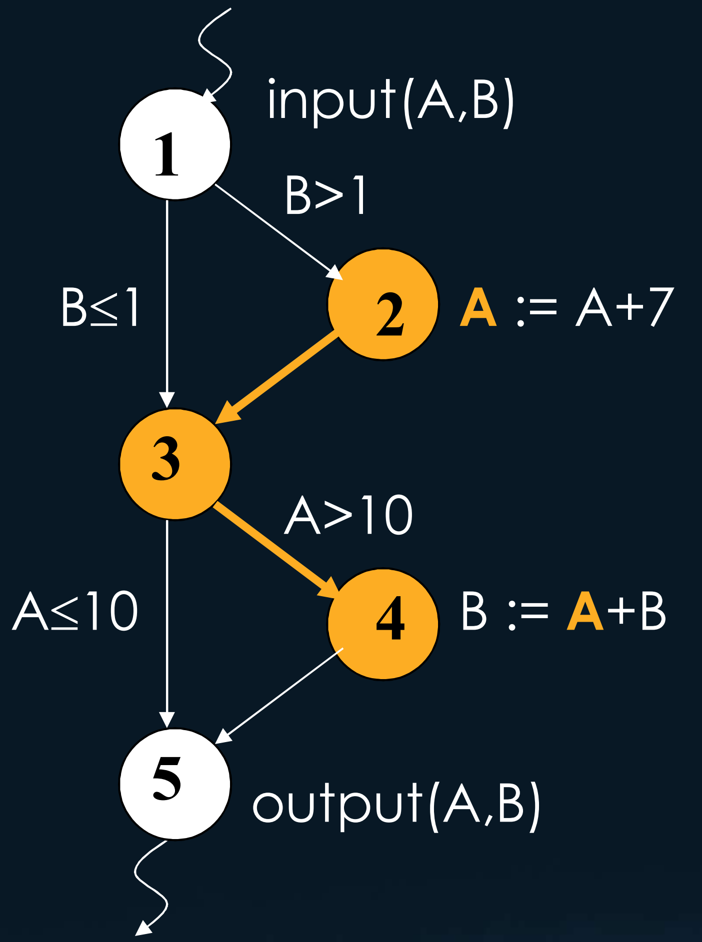
Identifying DU-Pairs – Variable A

du-pair	**path(s)**
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	**<1,3,5>**
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



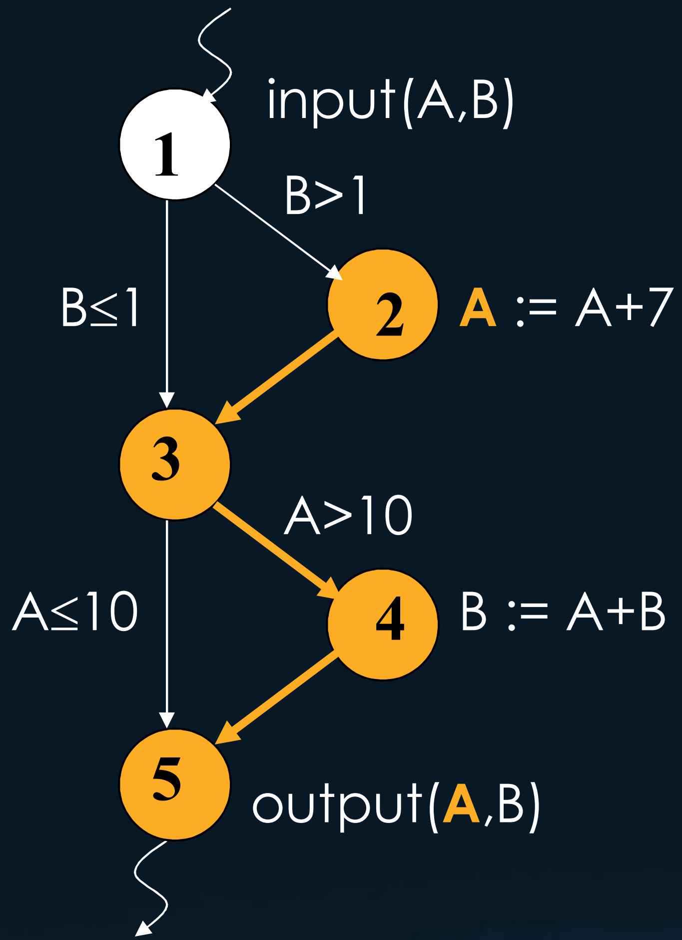
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	**<2,3,4>**
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



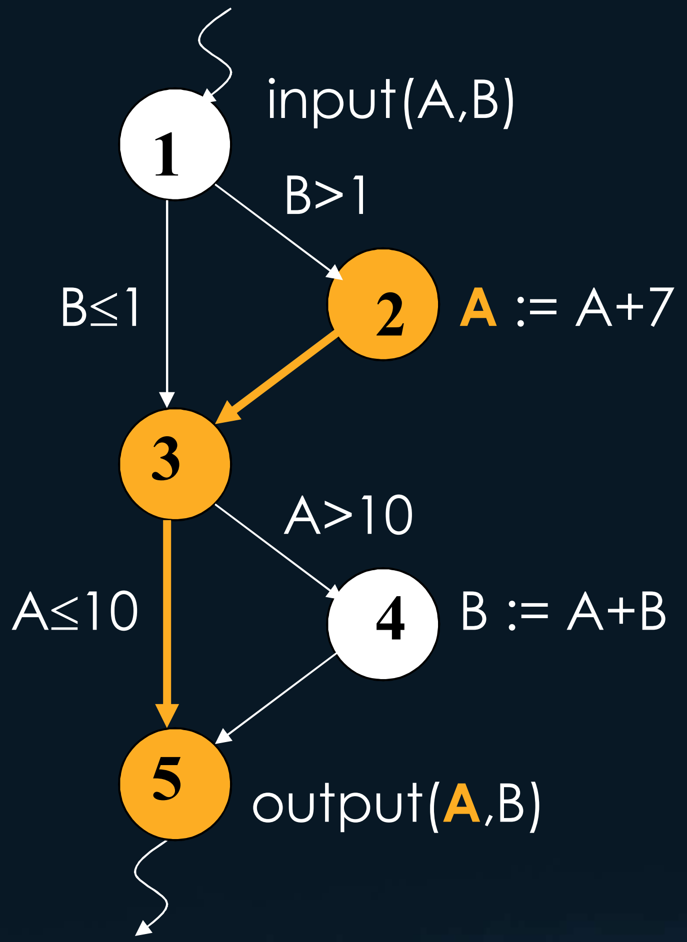
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	**<2,3,4,5>**
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



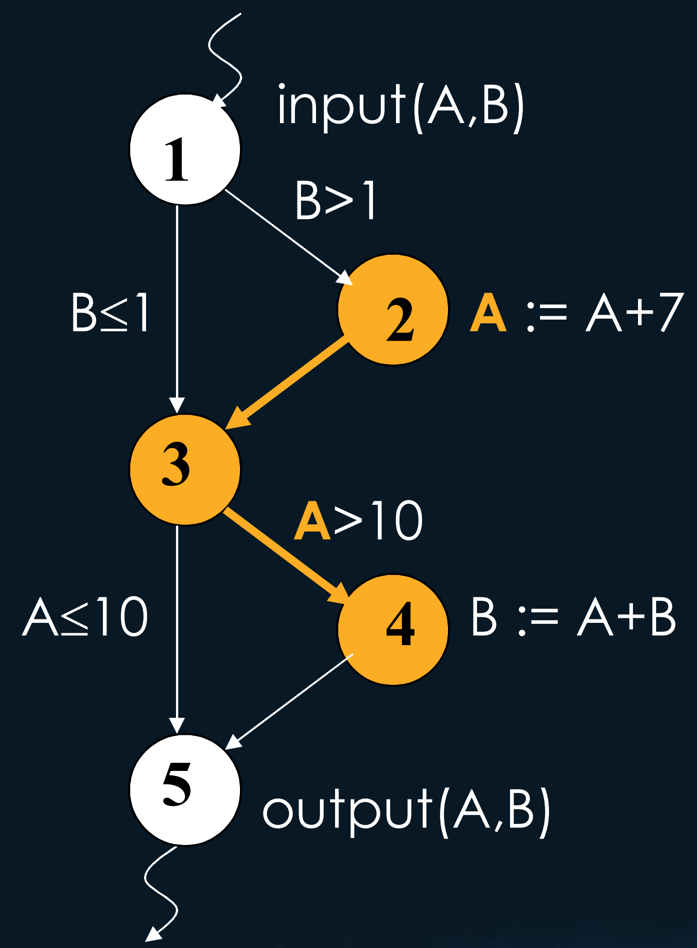
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	<2,3,5>



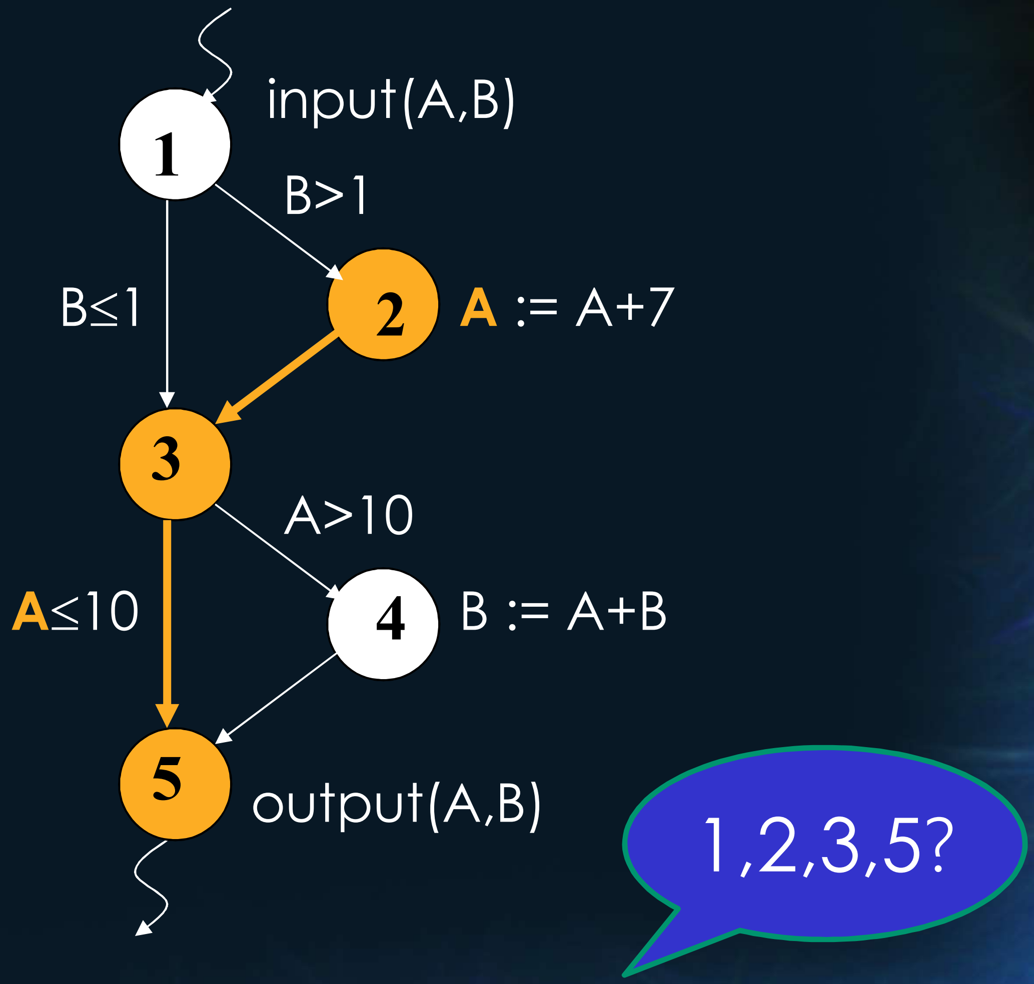
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	**<2,3,4>**
(2,<3,5>)	<2,3,5>



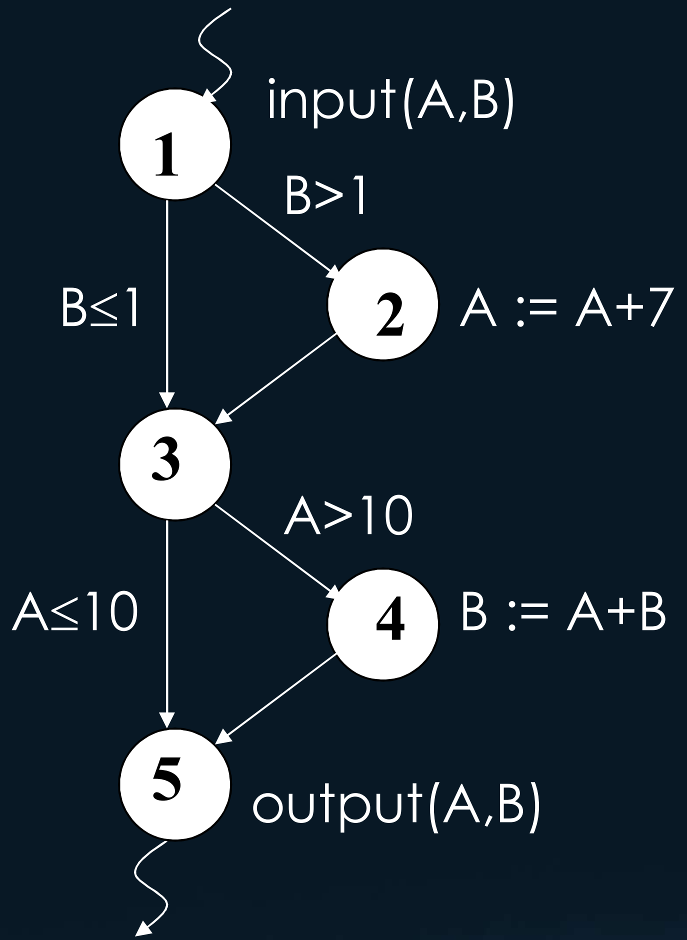
Identifying DU-Pairs – Variable A

du-pair	path(s)
(1,2)	<1,2>
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4>
(2,5)	<2,3,4,5>
	<2,3,5>
(2,<3,4>)	<2,3,4>
(2,<3,5>)	**<2,3,5>**



Identifying DU-Pairs – Variable B

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,3,4>
(1,5)	<1,2,3,5>
	<1,3,5>
(1,<1,2>)	<1,2>
(1,<1,3>)	<1,3>
(4,5)	<4,5>



Dataflow Test Coverage Criteria

- ***All-Defs:*** for **every program variable v**, **at least one def-clear path** from **every definition** of v to **at least one c-use or one p-use** of v must be covered.

Dataflow Test Coverage Criteria (cont'd)

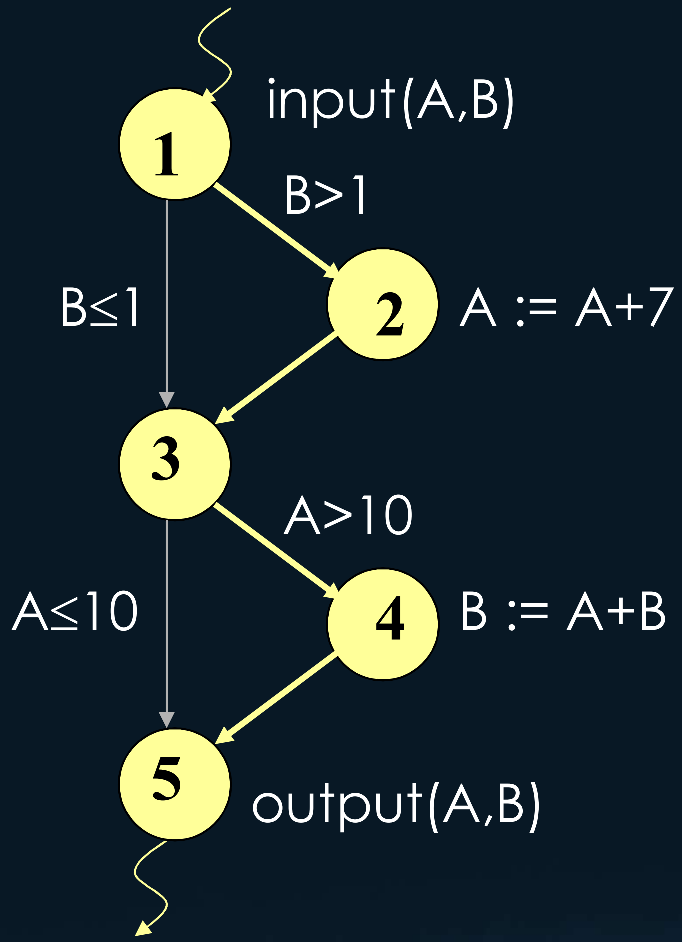
- Consider a test case executing path:

   1. **<1,2,3,4,5>**

- Identify all def-clear paths covered (i.e., subsumed) by this path for each variable.
- Are all definitions for each variable associated with at least one of the subsumed def-clear paths?

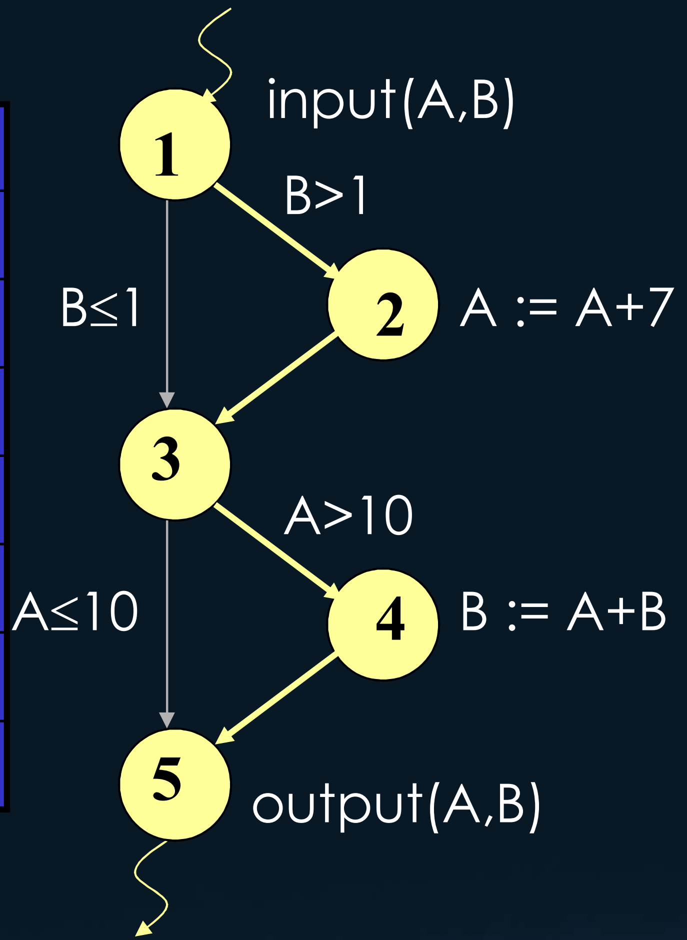
Def-Clear Paths Subsumed by <1,2,3,4,5> for Variable A

du-pair	path(s)
(1,2)	<1,2> ✓
(1,4)	<1,3,4>
(1,5)	<1,3,4,5>
	<1,3,5>
(1,<3,4>)	<1,3,4>
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4> ✓
(2,5)	<2,3,4,5> ✓
	<2,3,5>
(2,<3,4>)	<2,3,4> ✓
(2,<3,5>)	<2,3,5>



Def-Clear Paths Subsumed by <1,2,3,4,5> for Variable B

du-pair	path(s)
(1,4)	<1,2,3,4> ✓
	<1,3,4>
(1,5)	<1,2,3,5>
	<1,3,5>
(4,5)	<4,5> ✓
(1,<1,2>)	<1,2> ✓
(1,<1,3>)	<1,3>



Dataflow Test Coverage Criteria (cont'd)

- Since **<1,2,3,4,5>** covers at least one def-clear path from every definition of A/B to at least one c-use or p-use of A/B, All-Defs coverage is achieved.

Dataflow Test Coverage Criteria (cont'd)

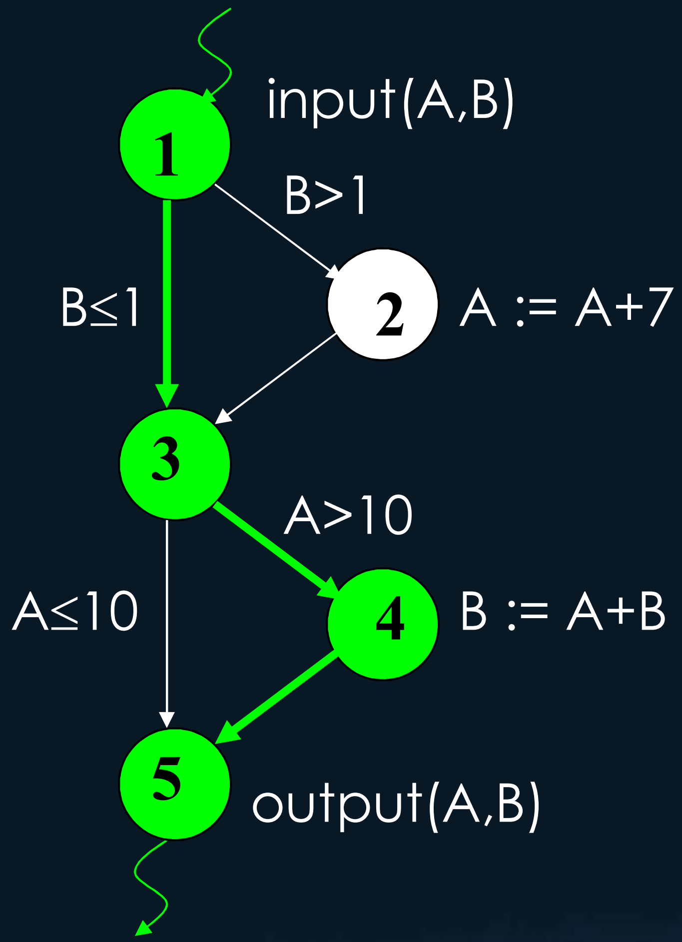
- ***All-Uses:*** for **every program variable v, at least one def-clear path** from **every definition** of v to **every c-use** and **every p-use** of v must be covered.
- Consider additional test cases executing paths:

  2. **<1,3,4,5>**
  3. **<1,2,3,5>**

- Do all three test cases provide All-Uses coverage?

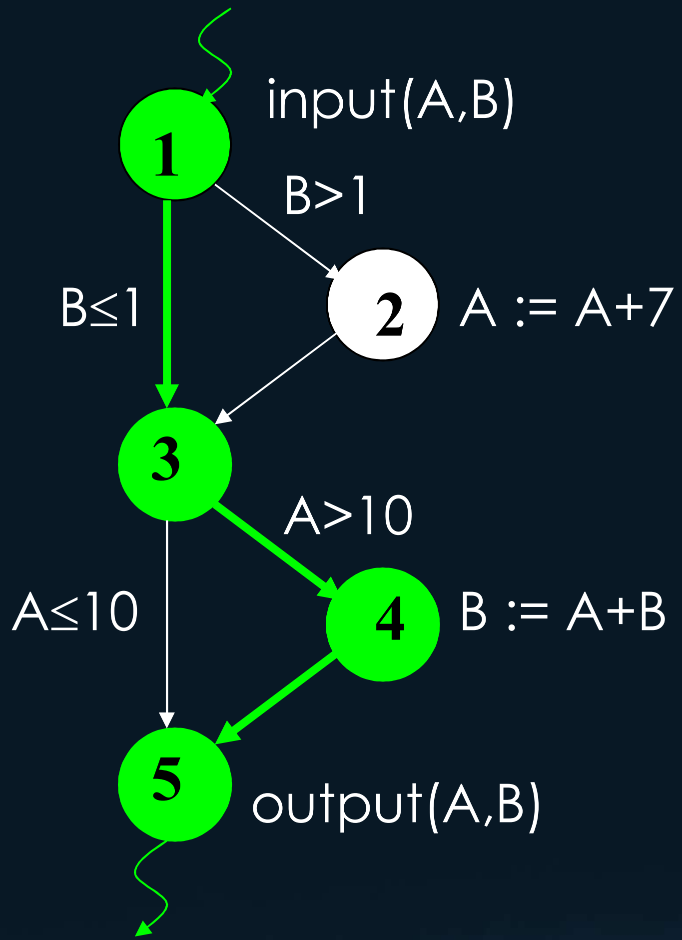
Def-Clear Paths Subsumed by <1,3,4,5> for Variable A

du-pair	path(s)
(1,2)	<1,2> ✓
(1,4)	<1,3,4> ✓
(1,5)	<1,3,4,5> ✓
	<1,3,5>
(1,<3,4>)	<1,3,4> ✓
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4> ✓
(2,5)	<2,3,4,5> ✓
	<2,3,5>
(2,<3,4>)	<2,3,4> ✓
(2,<3,5>)	<2,3,5>



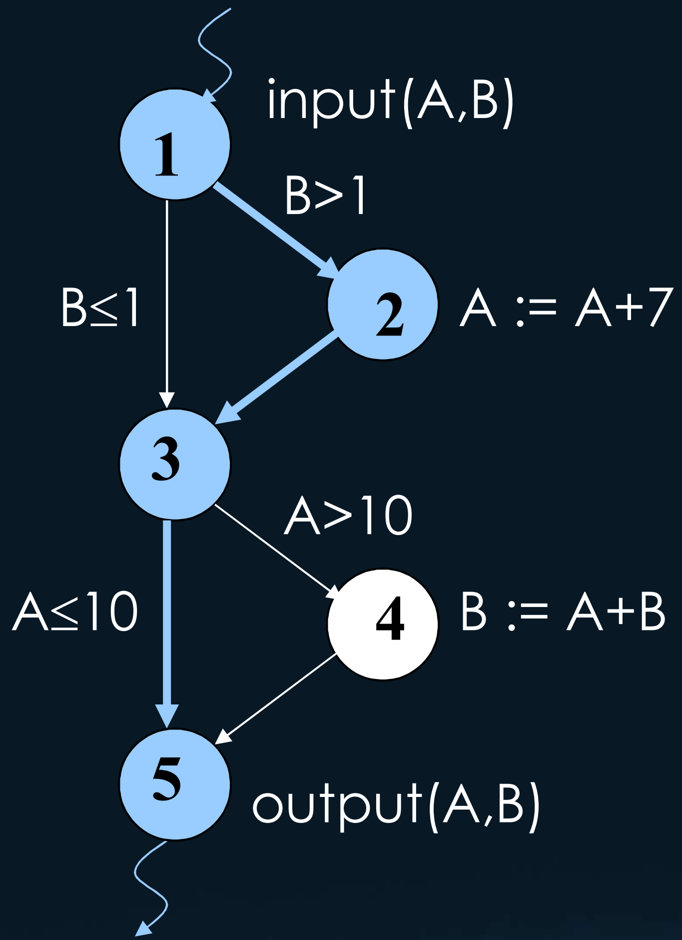
Def-Clear Paths Subsumed by <1,3,4,5> for Variable B

du-pair	path(s)
(1,4)	<1,2,3,4> ✓
	<1,3,4> ✓
(1,5)	<1,2,3,5>
	<1,3,5>
(4,5)	<4,5> ✓ ✓
(1,<1,2>)	<1,2> ✓
(1,<1,3>)	<1,3> ✓



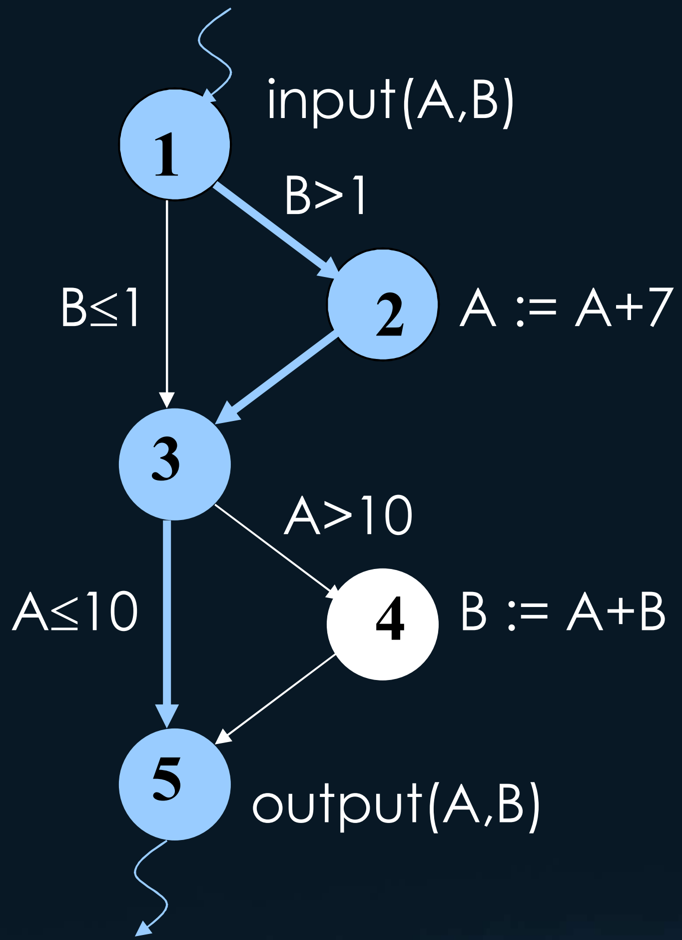
Def-Clear Paths Subsumed by <1,2,3,5> for Variable A

du-pair	path(s)
(1,2)	<1,2> ✓ ✓
(1,4)	<1,3,4> ✓
(1,5)	<1,3,4,5> ✓
	<1,3,5>
(1,<3,4>)	<1,3,4> ✓
(1,<3,5>)	<1,3,5>
(2,4)	<2,3,4> ✓
(2,5)	<2,3,4,5> ✓
	<2,3,5> ✓
(2,<3,4>)	<2,3,4> ✓
(2,<3,5>)	<2,3,5> ✓



Def-Clear Paths Subsumed by <1,2,3,5> for Variable B

du-pair	path(s)
(1,4)	<1,2,3,4> ✓
	<1,3,4> ✓
(1,5)	<1,2,3,5> ✓
	<1,3,5>
(4,5)	<4,5> ✓ ✓
(1,<1,2>)	<1,2> ✓ ✓
(1,<1,3>)	<1,3> ✓



Dataflow Test Coverage Criteria (cont’d)

- Since none of the three test cases covers the du-pair (1,<3,5>) for variable A, All-Uses Coverage is not provided.

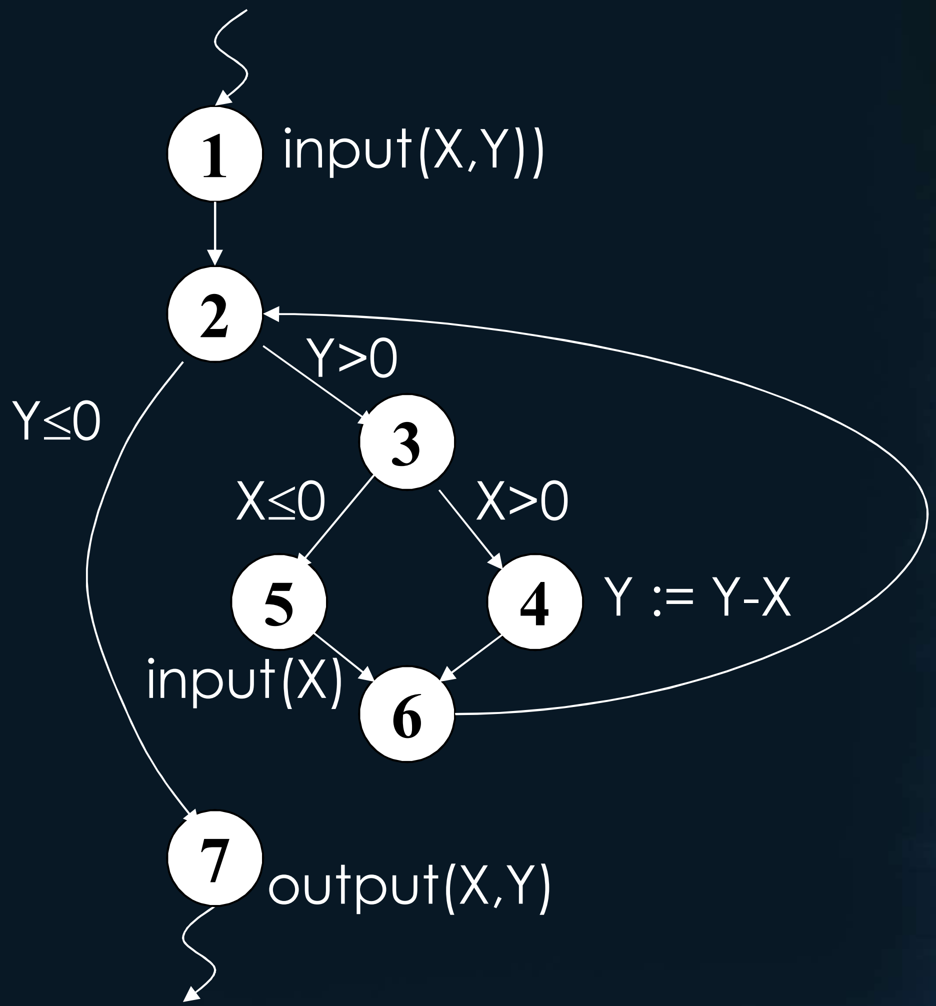
What would be even stronger than *All-Uses*?

- We have considered the *All-Defs* and the *All-Uses* criteria so far. How could the definition of *All-Uses* be modified to make it even stronger? Recall:

    All-Uses: for every program variable v, **at least one def-clear path** from every definition of v to every c-use and every p-use of v must be covered.

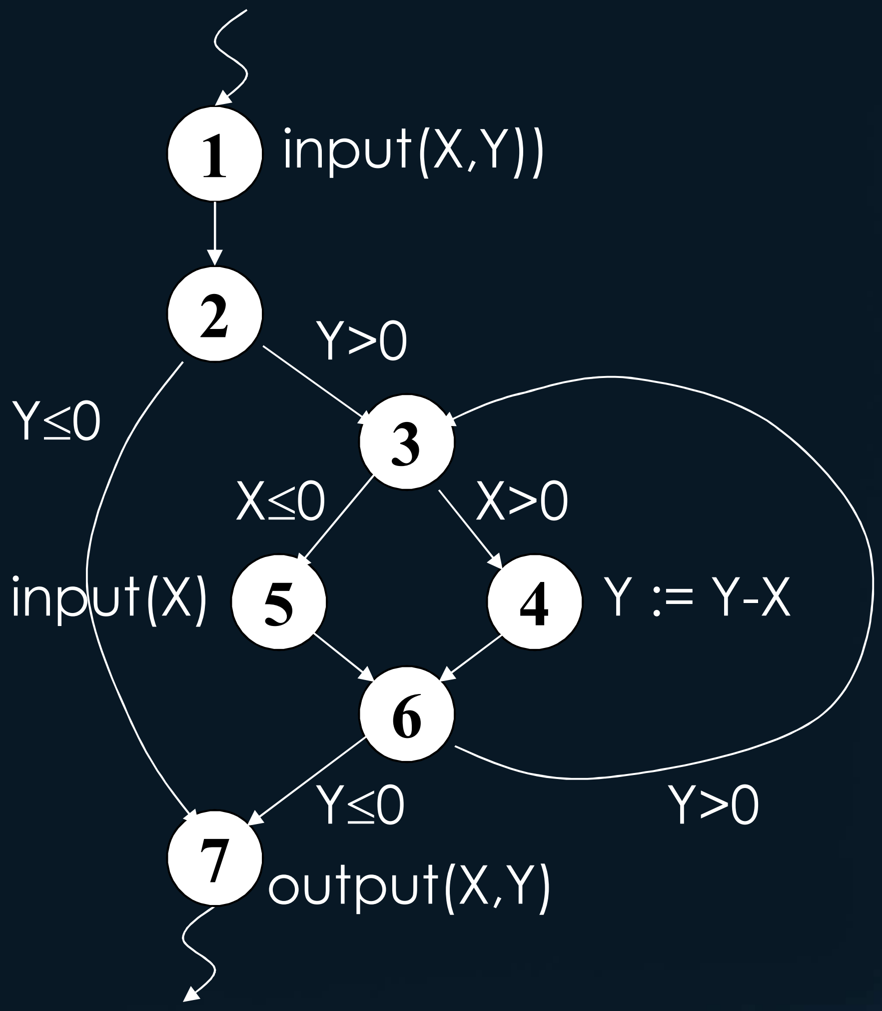
Exercise

1. input(X,Y)
2. while (Y>0) do
3.   if (X>0) then
4.   Y := Y-X
       else
5.   input(X)
       end_if_then_else
6. end_while
7. output(X,Y)



Exercise

1. input(X,Y)
2. while (Y>0) do
3. if (X>0) then
4. Y := Y-X
   else
5. input(X)
   end_if_then_else
6. end_while
7. output(X,Y)

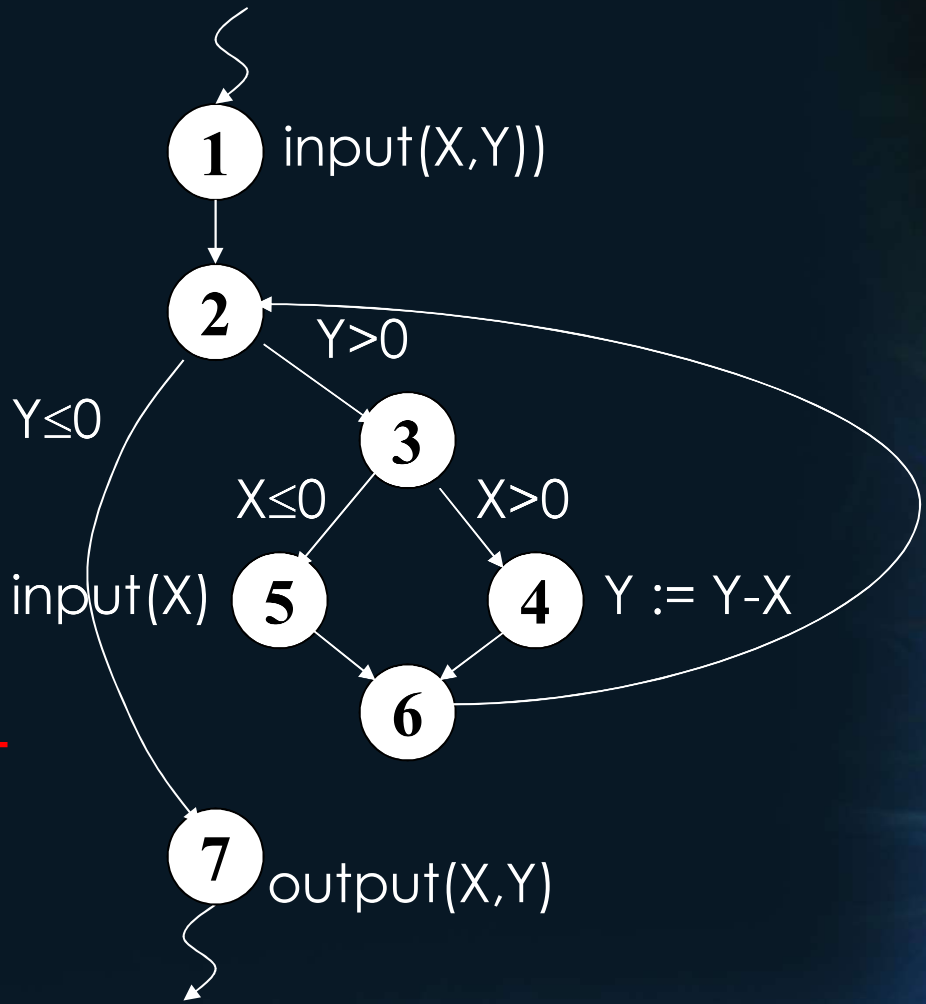


Exercise

- Identify DU-Pairs for variable X.

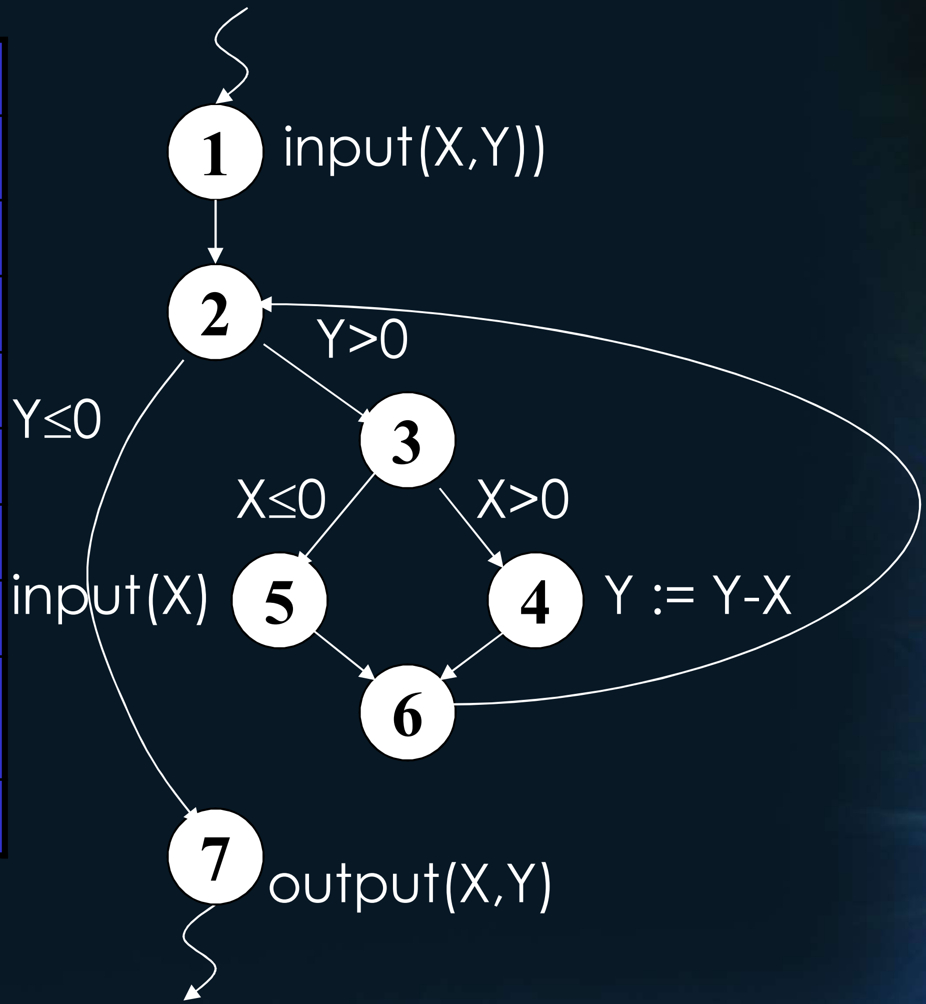
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,2,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,2,7>
	<1,2,3,4,6,(2,3,4,6)*,2,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,2,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,2,3,4>
	<5,6,2,3,4,(6,2,3,4)*>



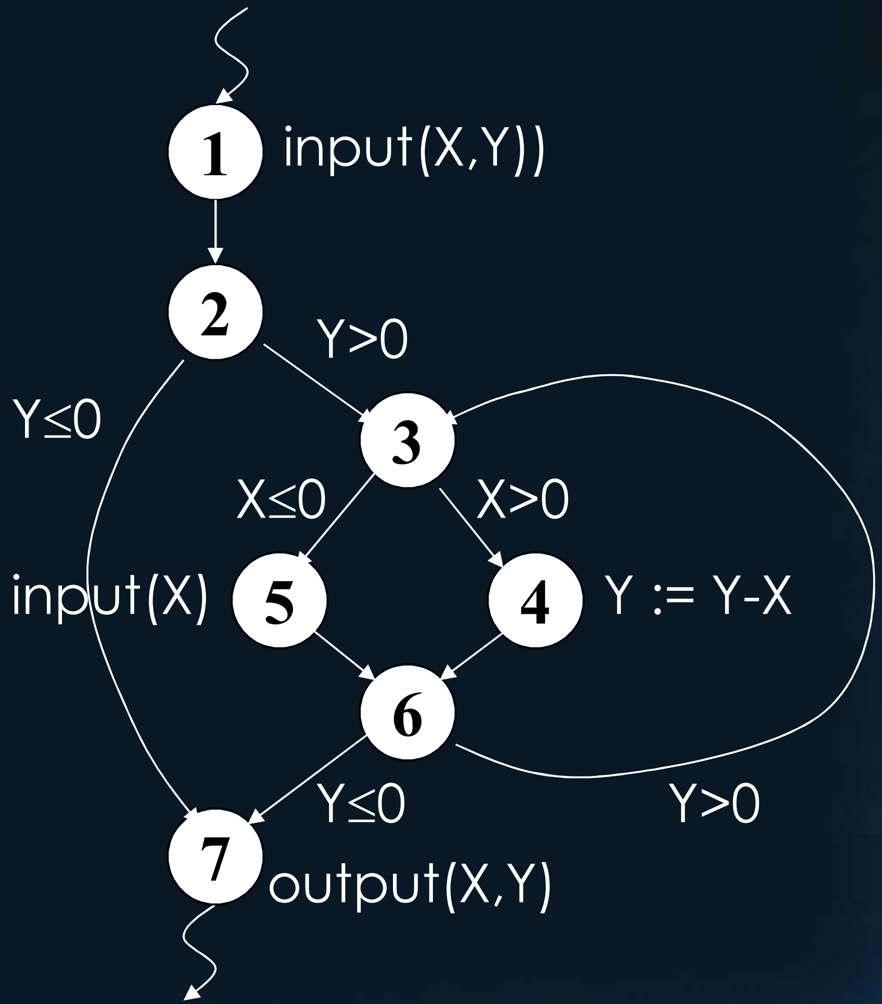
Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,2,7>†
	<5,6,2,3,4,6,2,7>
	<5,6,2,3,4,6,(2,3,4,6)*,2,7>
(5,<3,4>)	<5,6,2,3,4>
	<5,6,2,3,4,(6,2,3,4)*>
(5,<3,5>)	<5,6,2,3,5>
	<5,6,2,3,4,6,2,3,5>†
	<5,6,2,3,4,6,(2,3,4,6)*,2,3,5> †
† infeasible	



Identifying DU-Pairs – Variable X

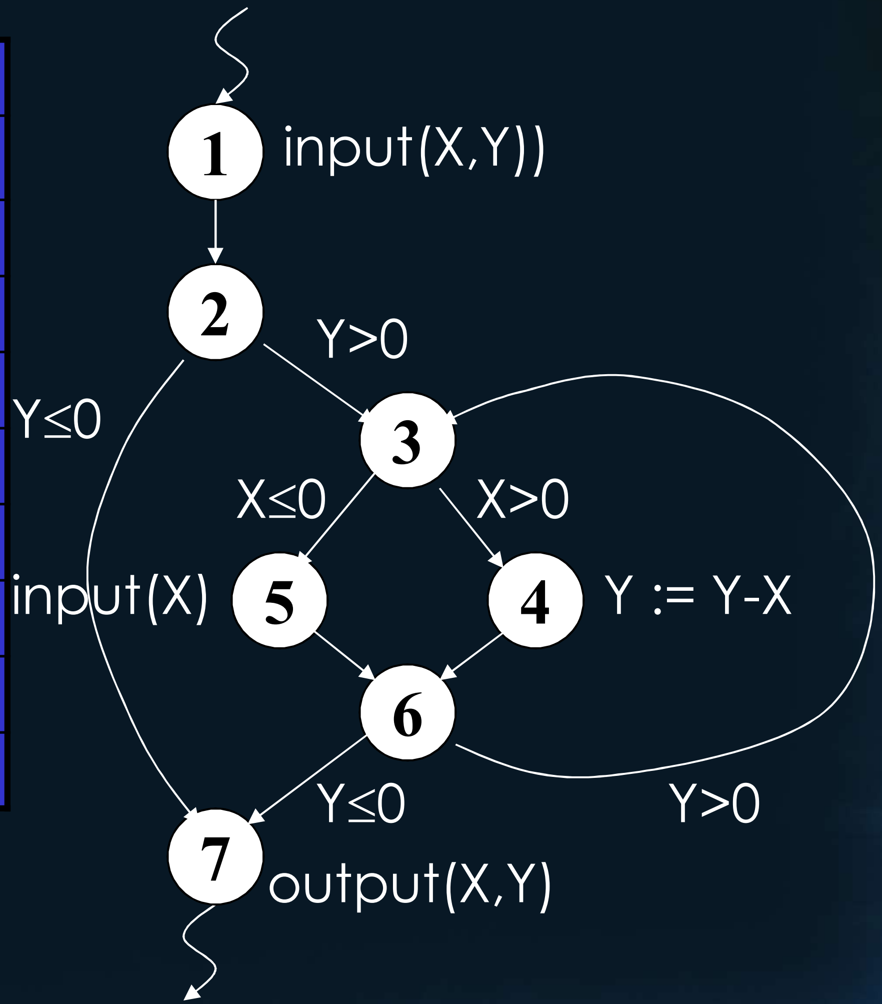
du-pair	**path(s)**
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



Identifying DU-Pairs – Variable X

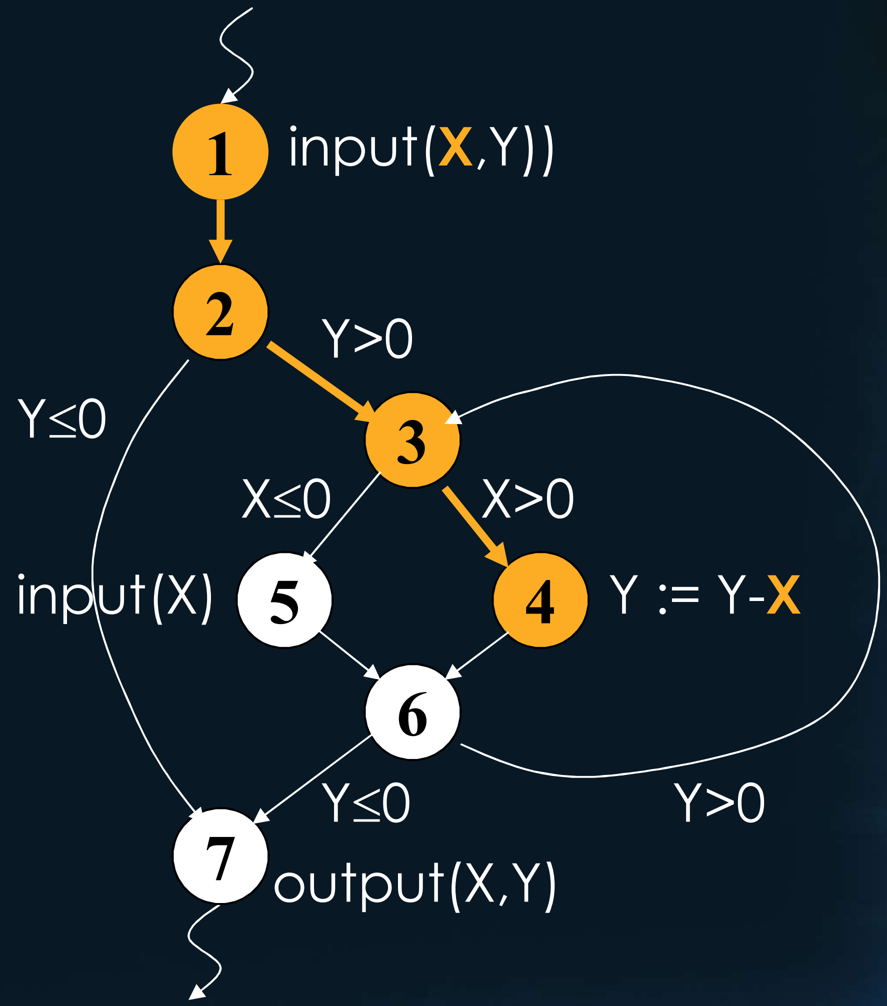
du-pair	path(s)
(5,7)	<5,6,7>†
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*,7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>†
	<5,6,3,4,6,(3,4,6)*,3,5>†

† infeasible



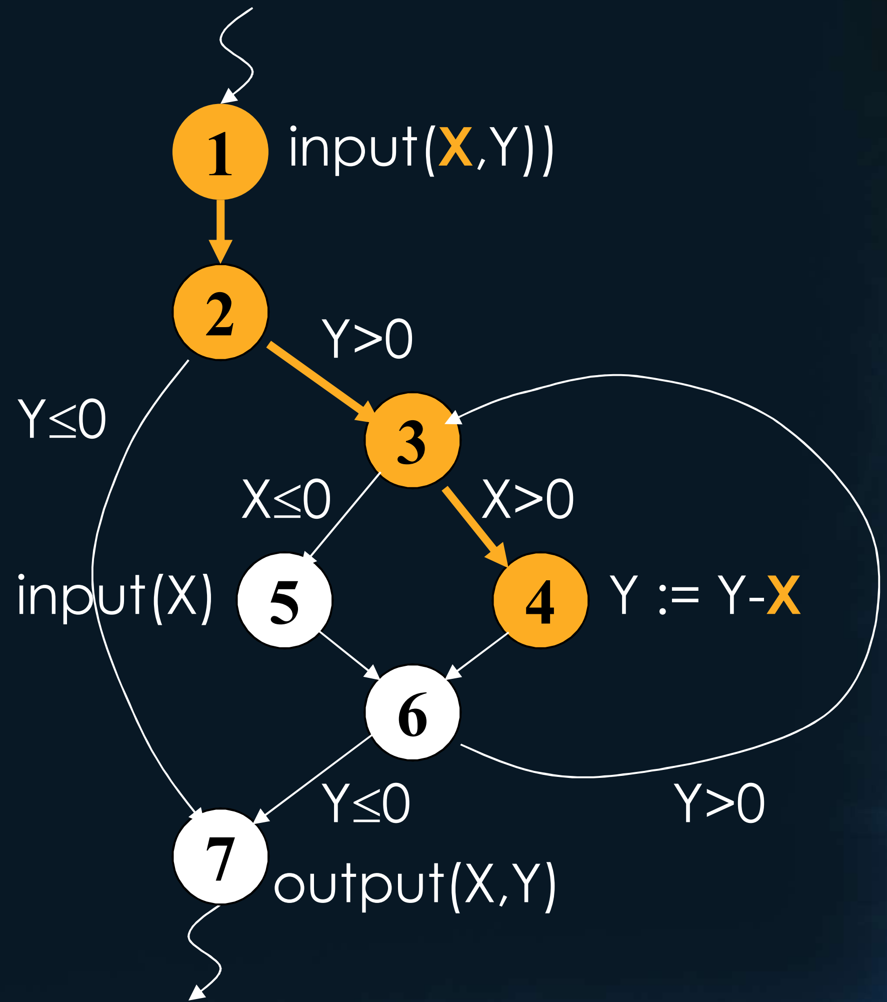
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	**<1,2,3,4>**
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



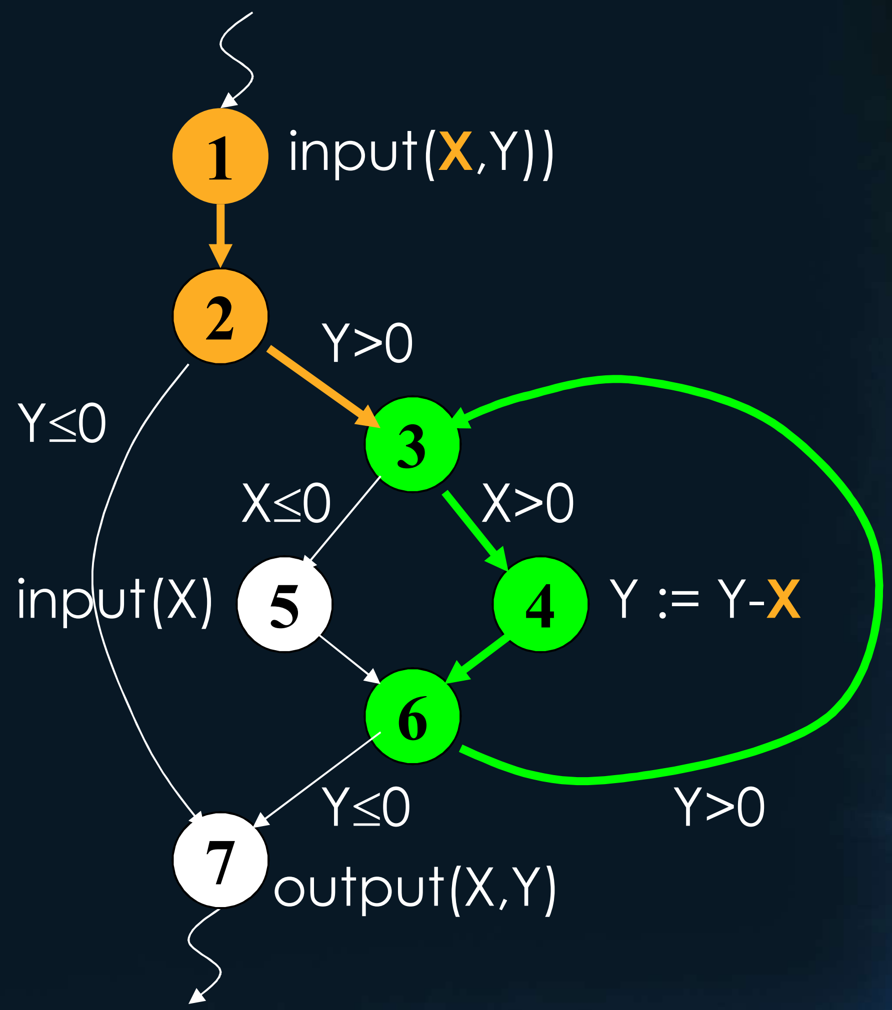
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



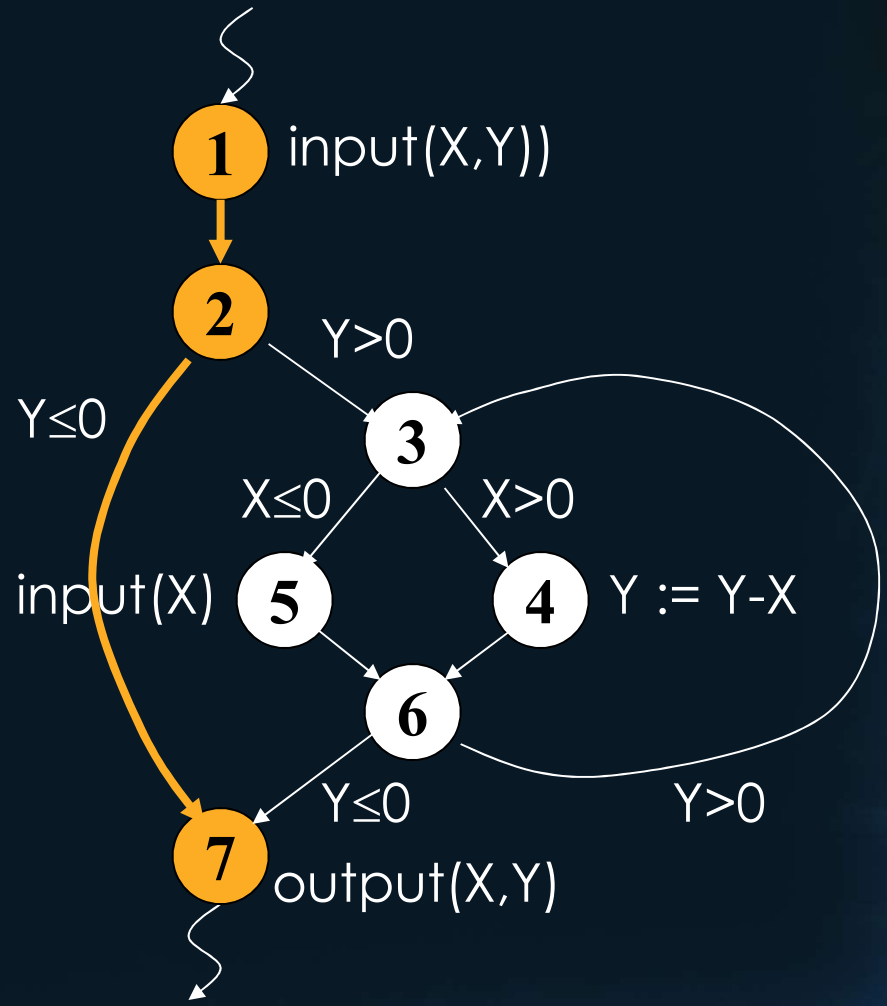
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



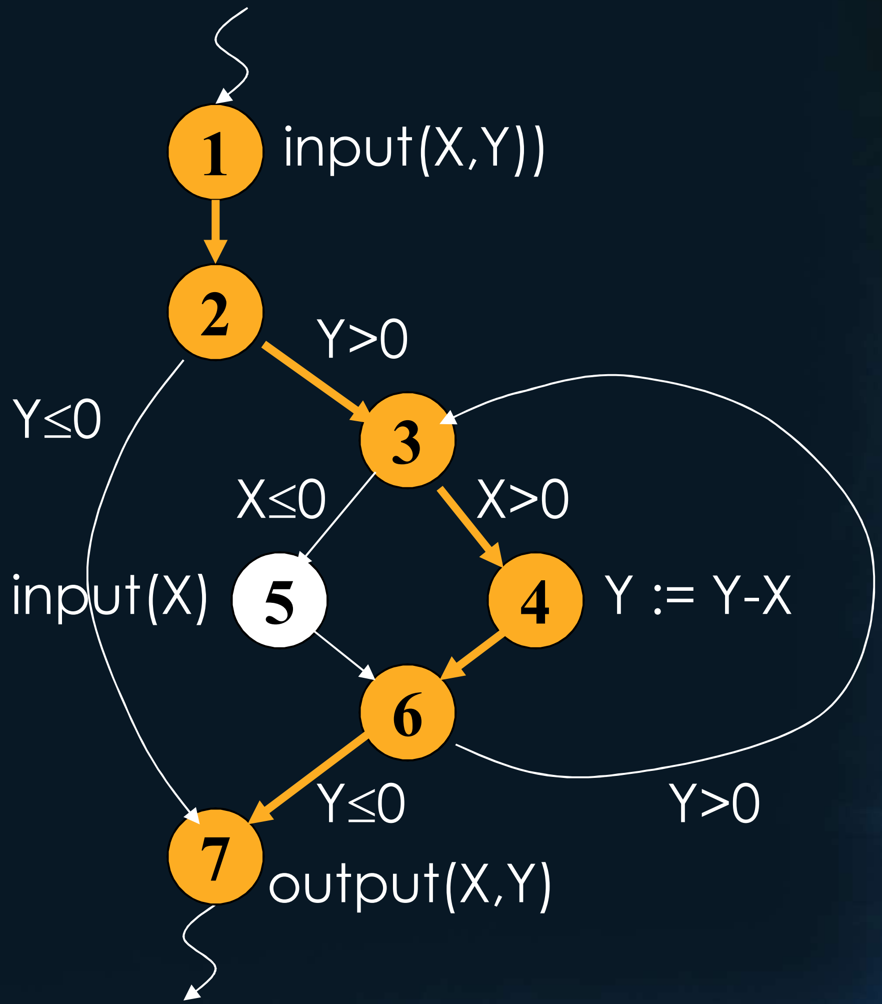
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	**<1,2,7>**
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



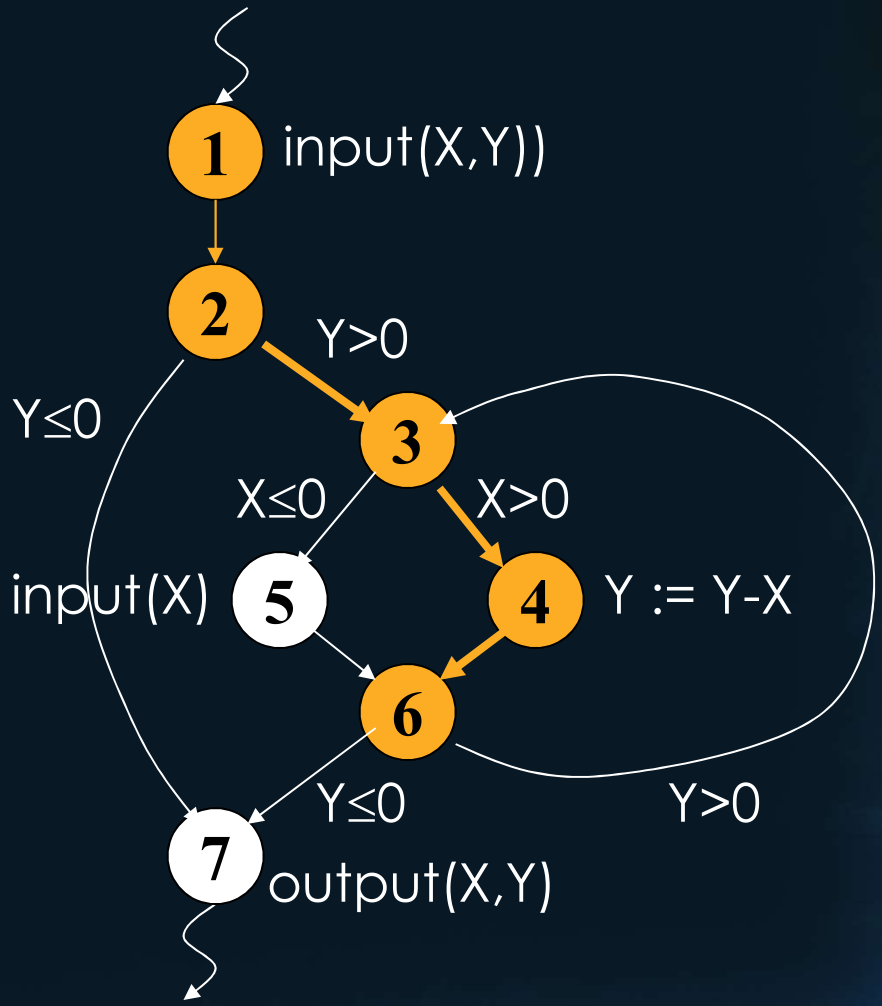
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



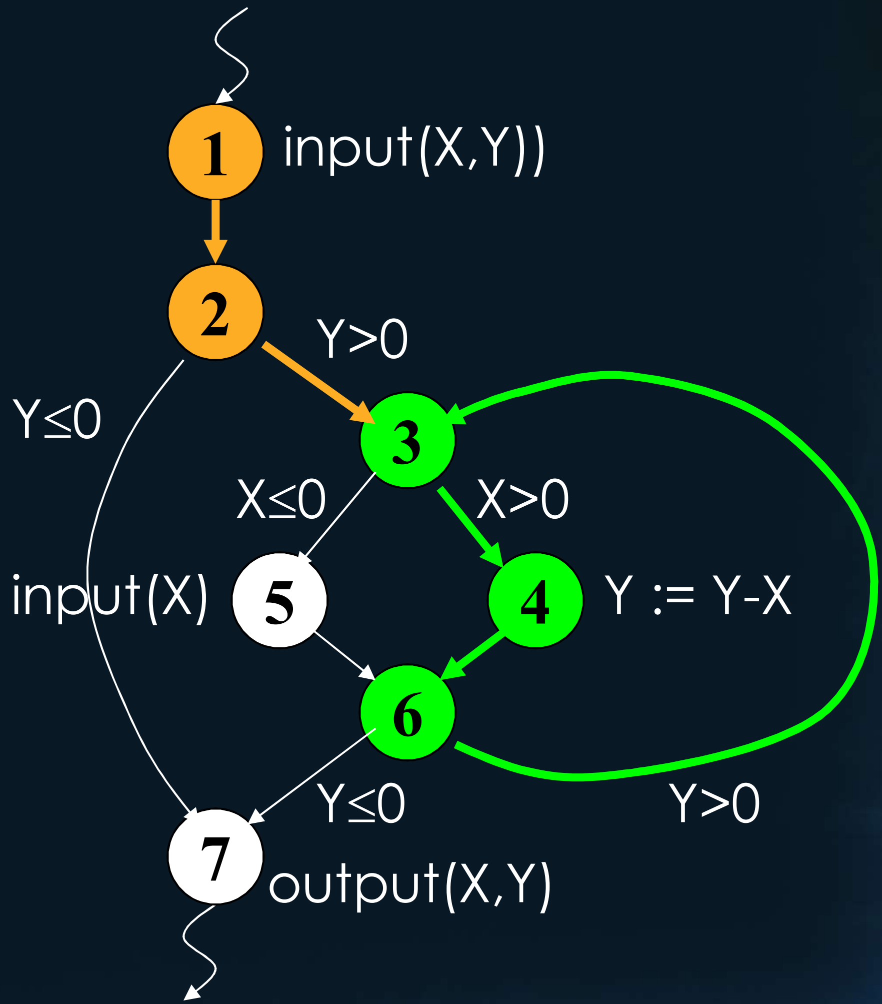
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



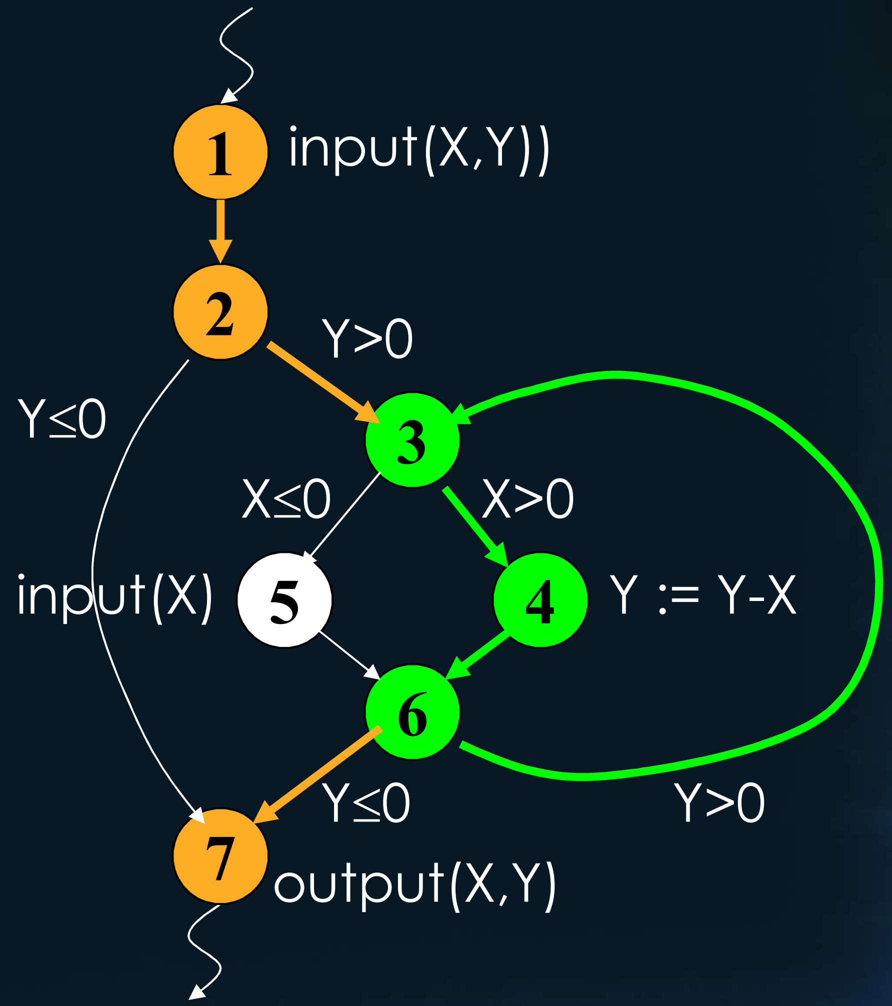
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



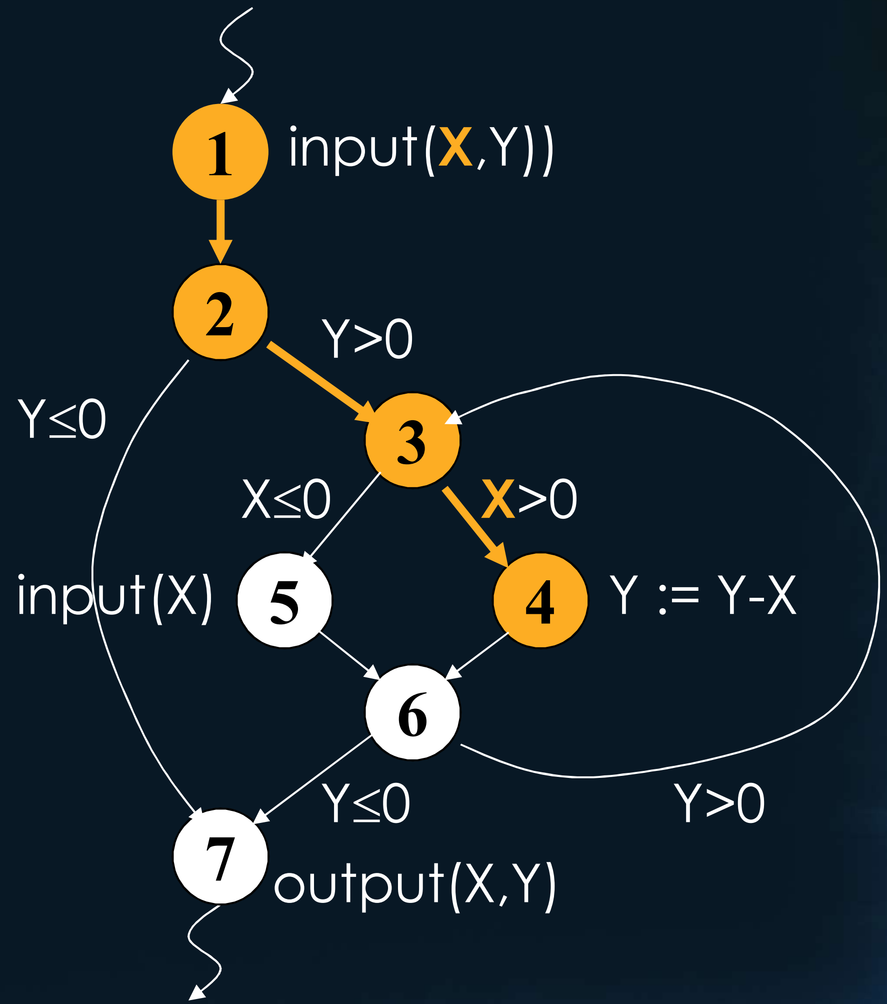
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



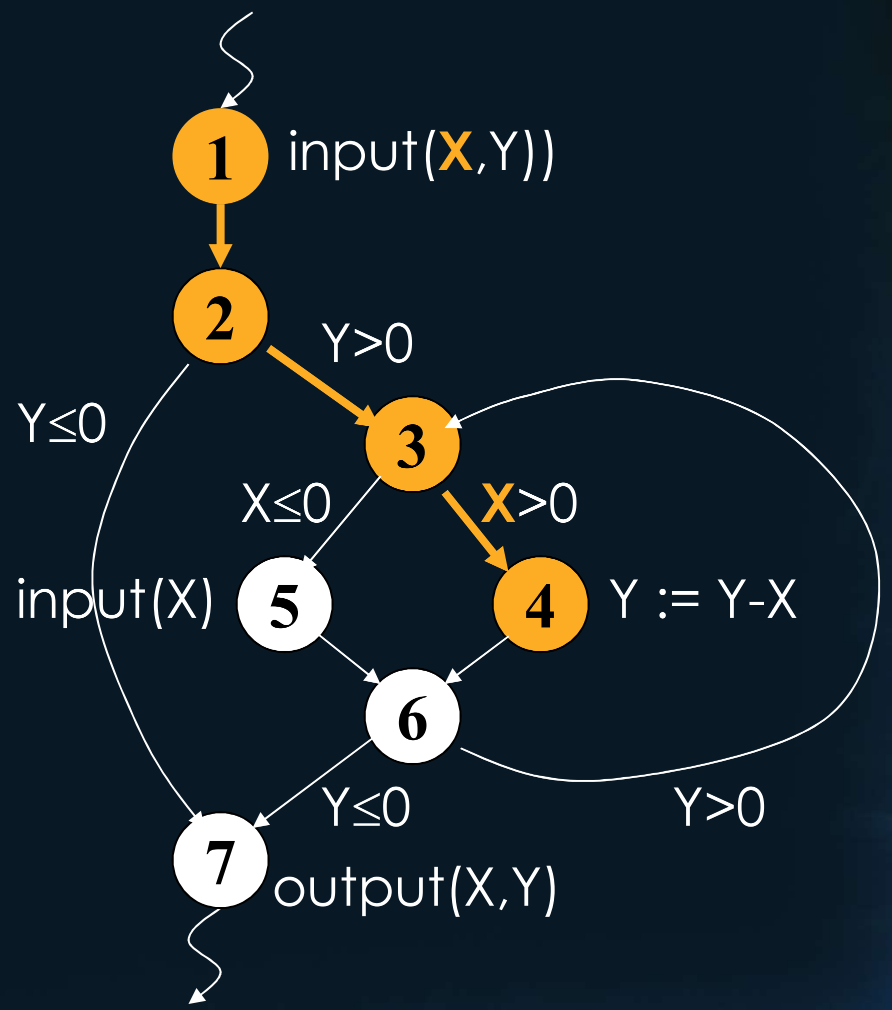
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	**<1,2,3,4>**
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



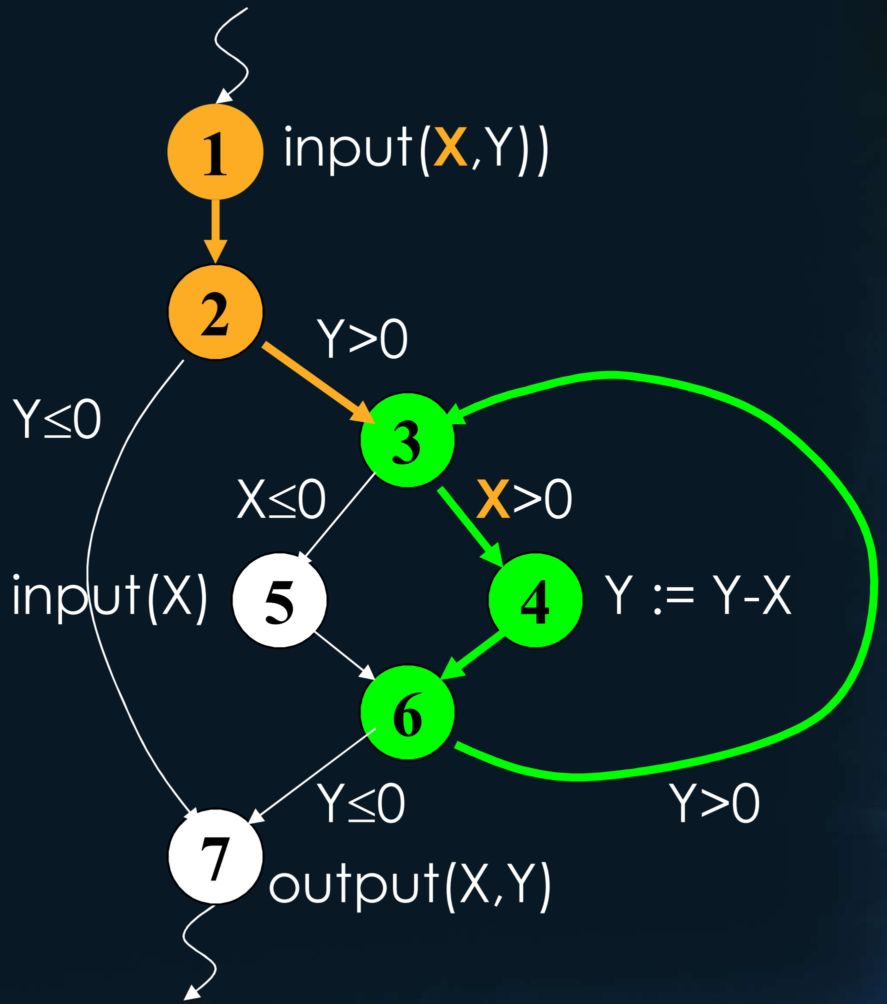
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



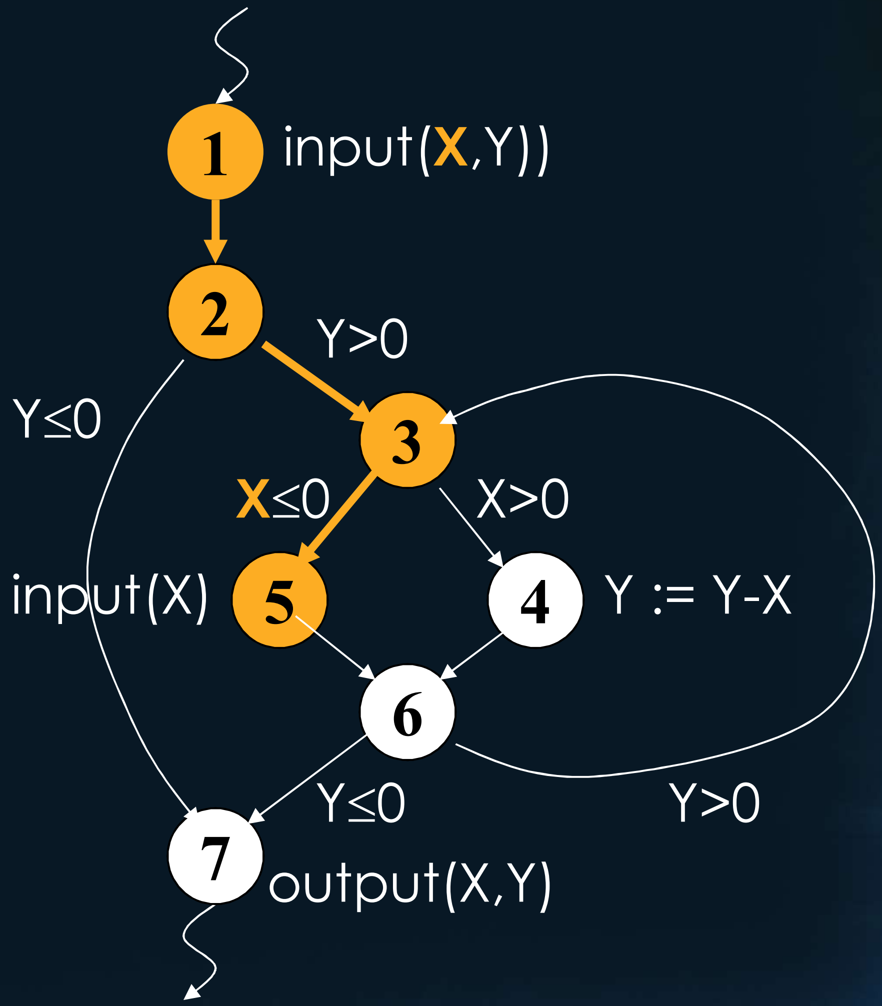
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



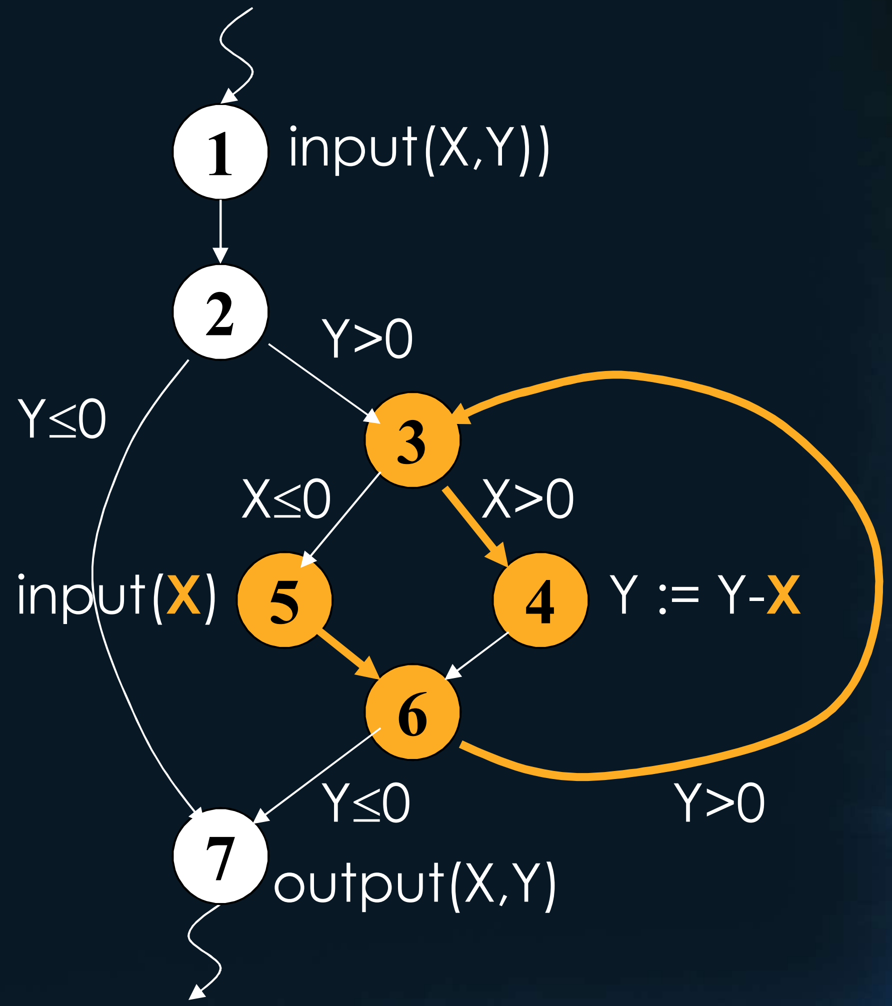
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	**<1,2,3,5>**
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



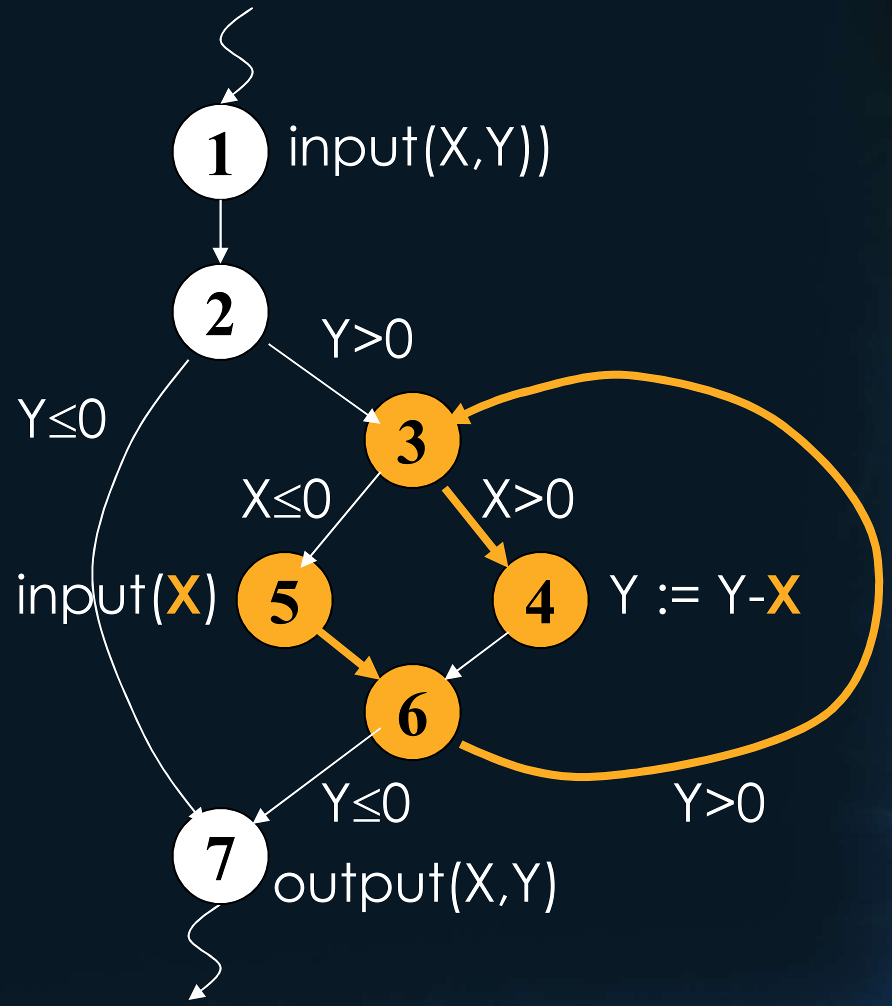
Identifying DU-Pairs – Variable X

du-pair	**path(s)**
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	**<5,6,3,4>**
	<5,6,3,4,(6,3,4)*>



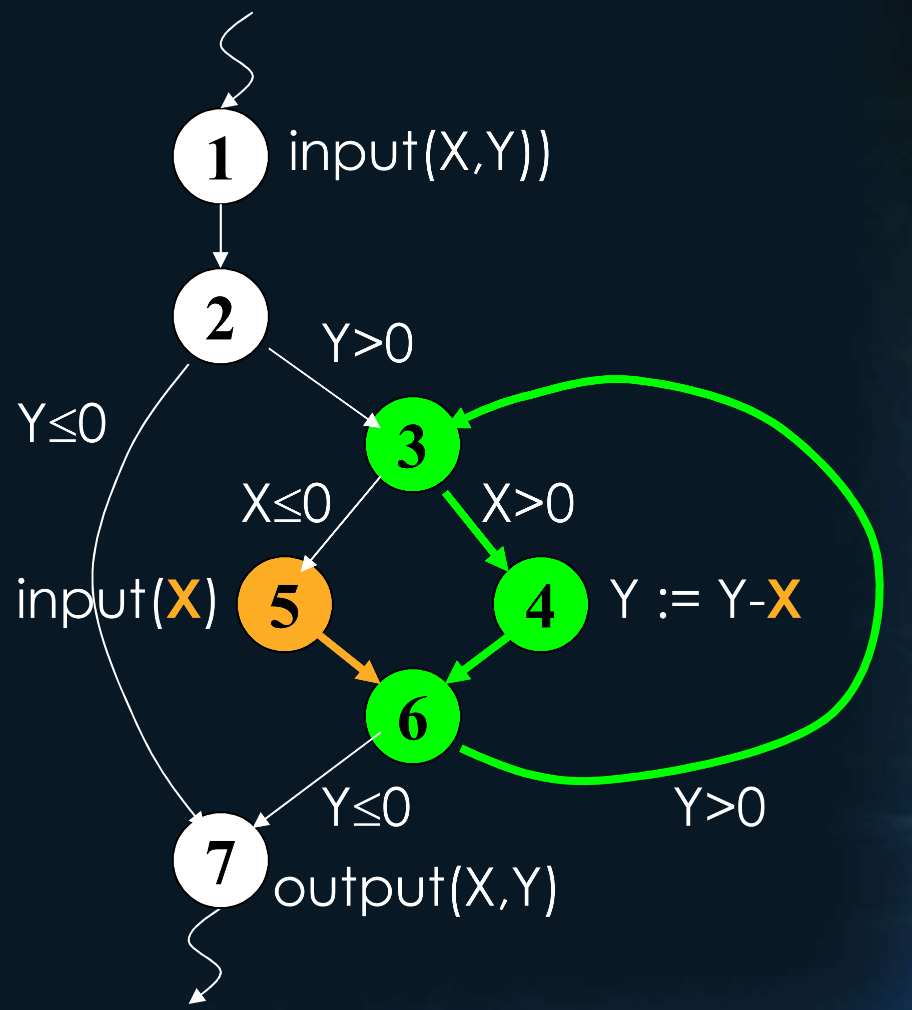
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



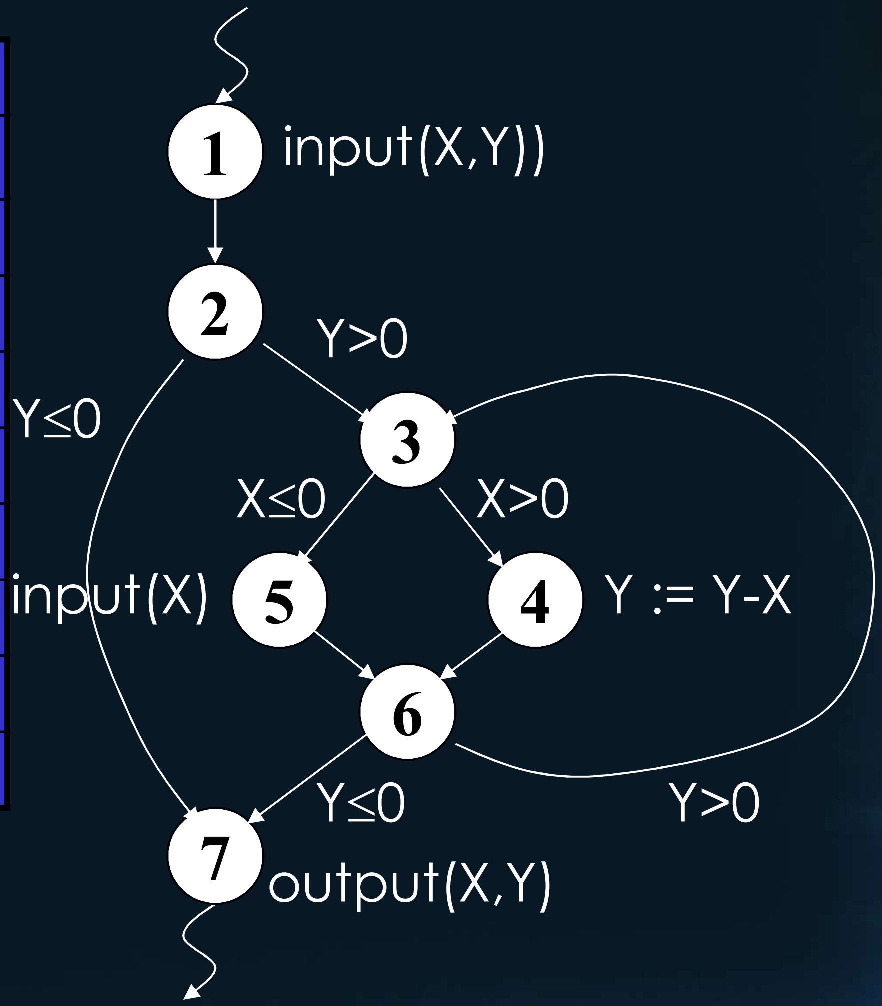
Identifying DU-Pairs – Variable X

du-pair	path(s)
(1,4)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,7)	<1,2,7>
	<1,2,3,4,6,7>
	<1,2,3,4,6,(3,4,6)*,7>
(1,<3,4>)	<1,2,3,4>
	<1,2,3,4,(6,3,4)*>
(1,<3,5>)	<1,2,3,5>
(5,4)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>



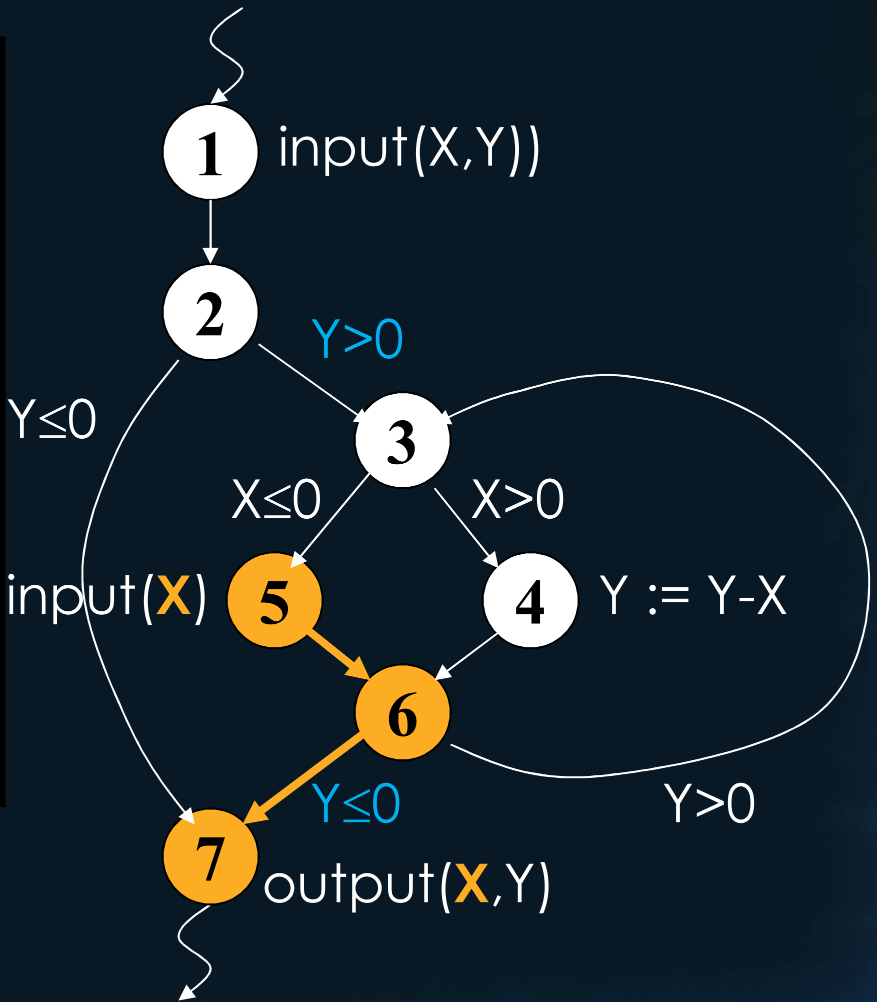
Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,7>
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*,7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>



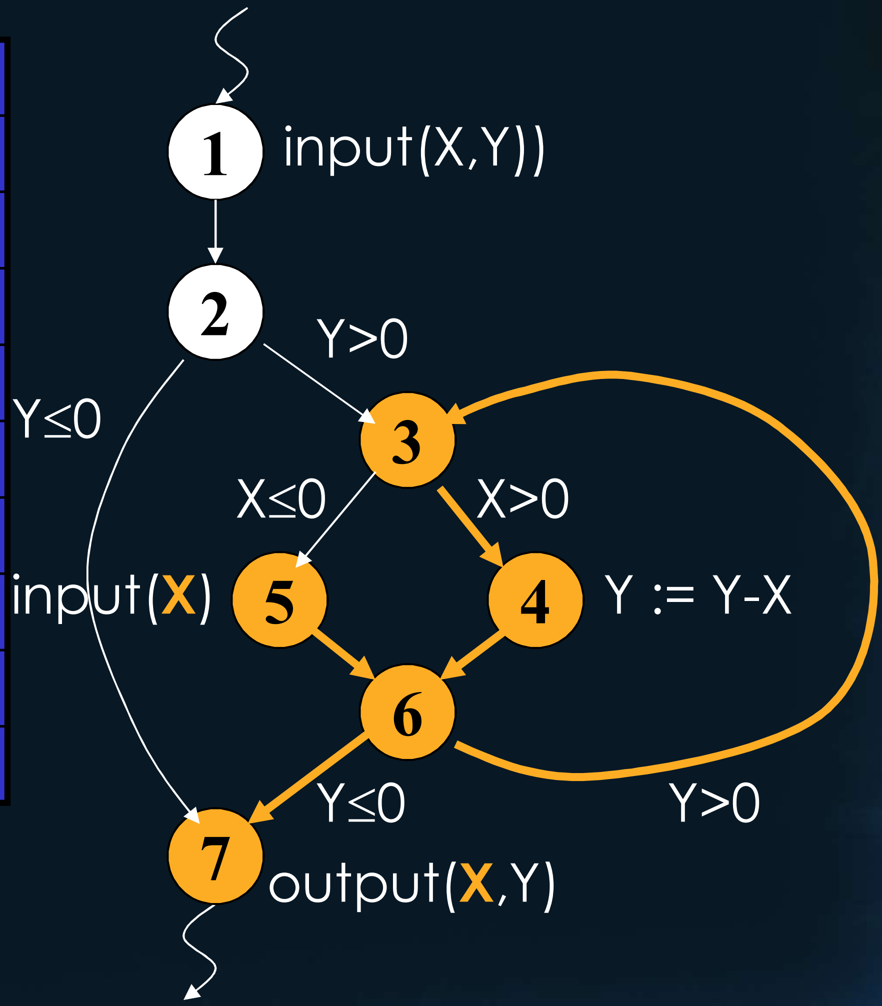
Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	**<5,6,7> †**
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*,7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>
† infeasible	



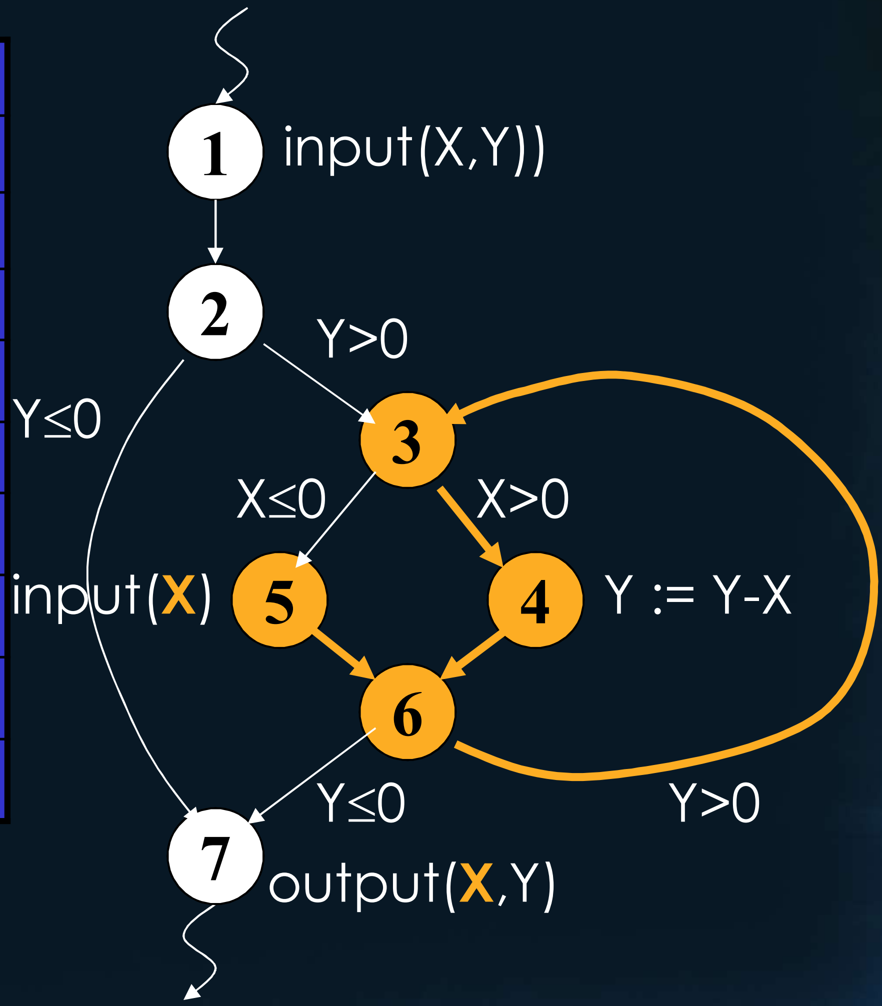
Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*,7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>
† infeasible	



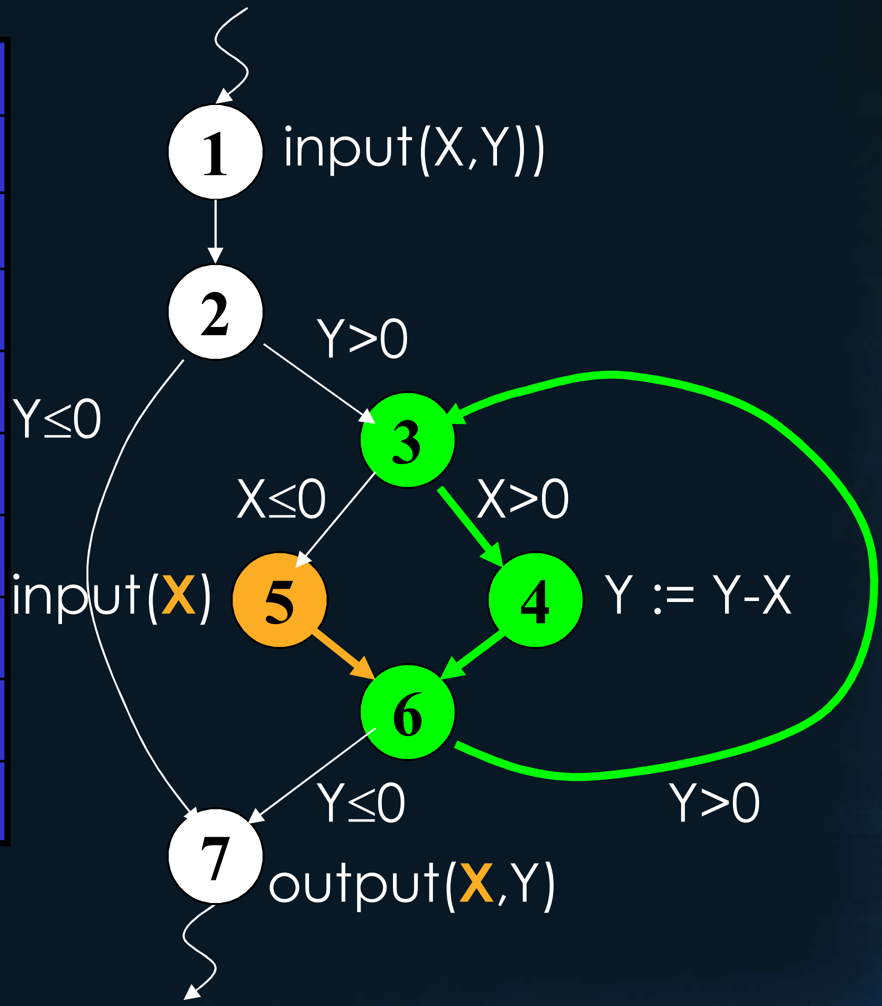
Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*,7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>
† infeasible	



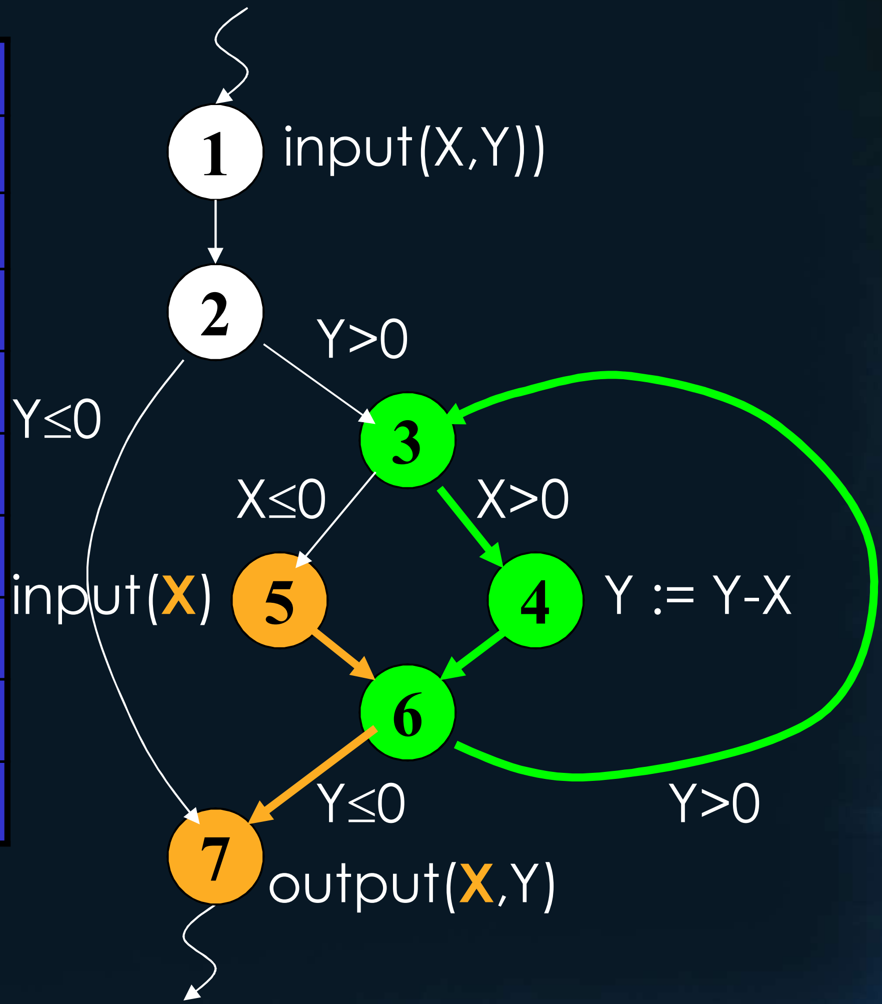
Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*,7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>
† infeasible	



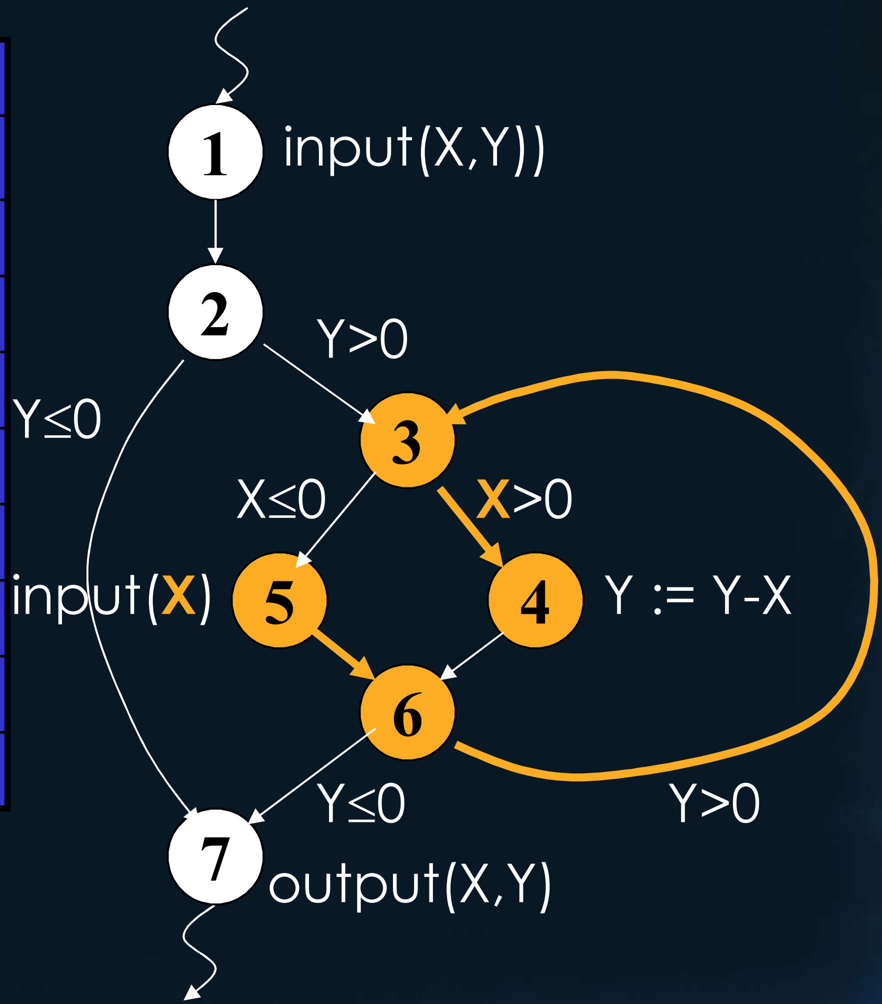
Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*,7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>
† infeasible	



Identifying DU-Pairs – Variable X

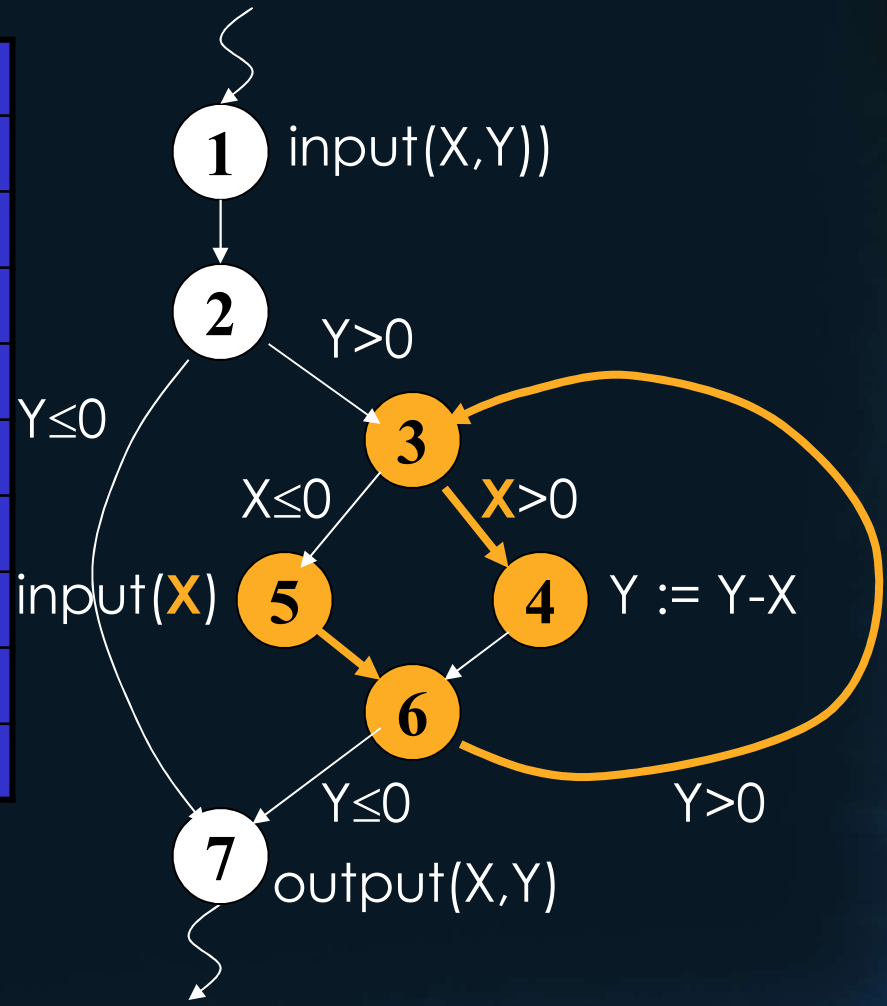
du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*,7>
(5,<3,4>)	**<5,6,3,4>**
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>
† infeasible	



Identifying DU-Pairs – Variable X

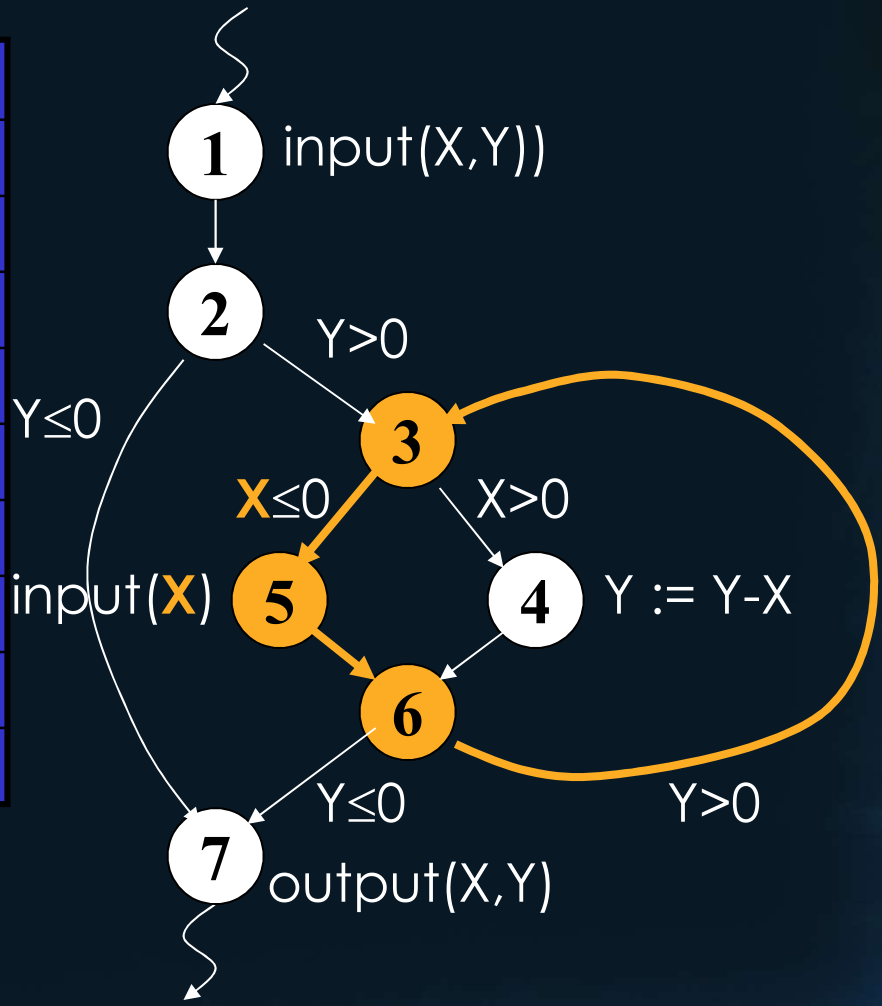
du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>

† infeasible



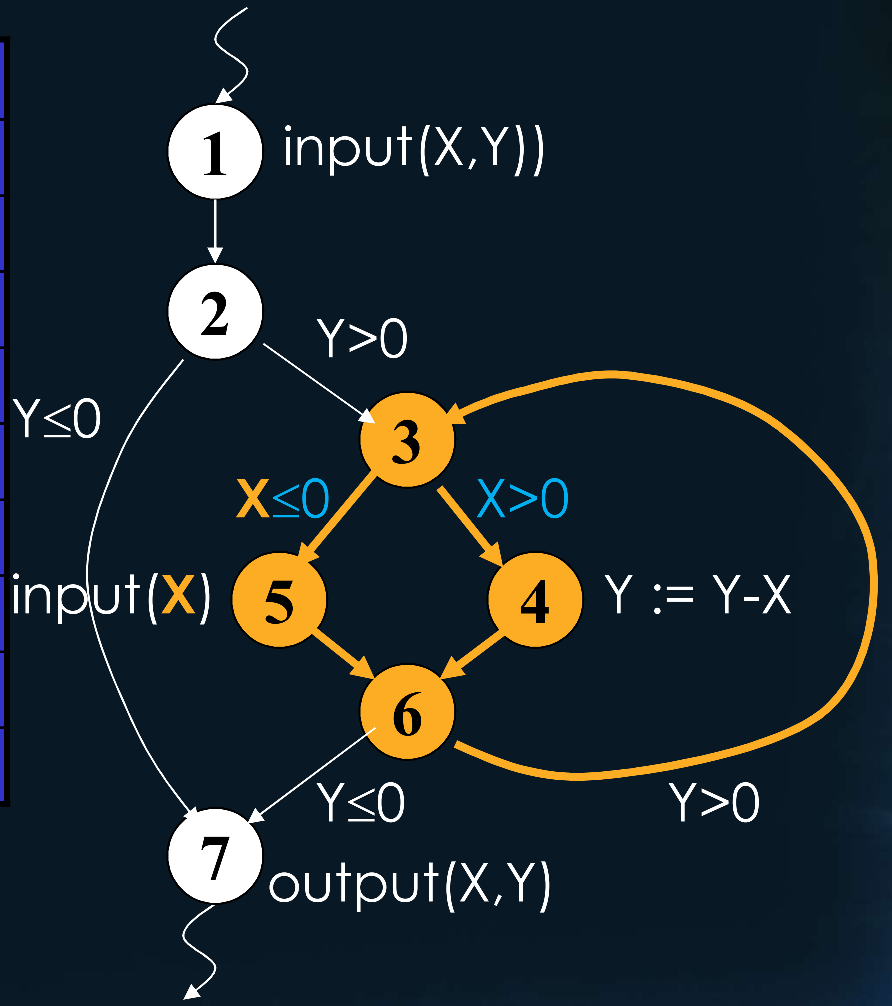
Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	**<5,6,3,5>**
	<5,6,3,4,6,3,5>
	<5,6,3,4,6,(3,4,6)*,3,5>
† infeasible	



Identifying DU-Pairs – Variable X

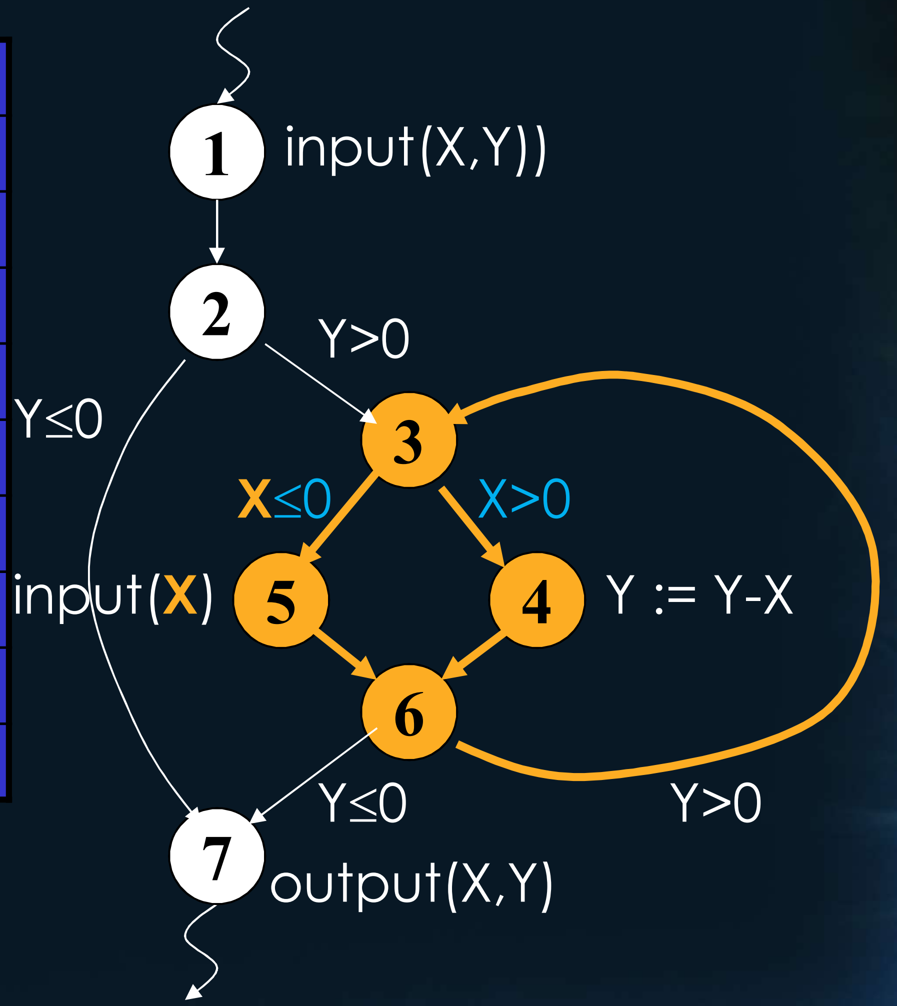
du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5> †
	<5,6,3,4,6,(3,4,6)*,3,5>
† **infeasible**	



Identifying DU-Pairs – Variable X

du-pair	path(s)
(5,7)	<5,6,7> †
	<5,6,3,4,6,7>
	<5,6,3,4,6,(3,4,6)*7>
(5,<3,4>)	<5,6,3,4>
	<5,6,3,4,(6,3,4)*>
(5,<3,5>)	<5,6,3,5>
	<5,6,3,4,6,3,5> †
	<5,6,3,4,6,(3,4,6)*,3,5> †

† infeasible



More Dataflow Terms and Definitions

- A path (either partial or complete) is **simple** if all edges within the path are distinct (i.e., different).
- A path is **loop-free** if all nodes within the path are distinct (i.e., different).

Simple and Loop-Free Paths

path	Simple?	Loop-free?
<1,3,4,2>	Yes	Yes
<1,2,3,2>	Yes	No
<1,2,3,1,2>	No	No
<1,2,3,2,4>	Yes	No

Simple and Loop-Free Paths (cont'd)

Which is *stronger*, **simple** or **loop-free**?

环路为1

无环路

More Dataflow Terms and Definitions

A path <n1,n2,...,nj,nk> is a **du-path** with respect to a variable v if v is defined at node n1 and either:

1. there is a **c-use** of v at node nk and <n1,n2,...,nj,nk> is a def-clear **simple** path,

or

1. there is a **p-use** of v at edge <nj,nk> and <n1,n2,...nj> is a def-clear **loop-free** path.

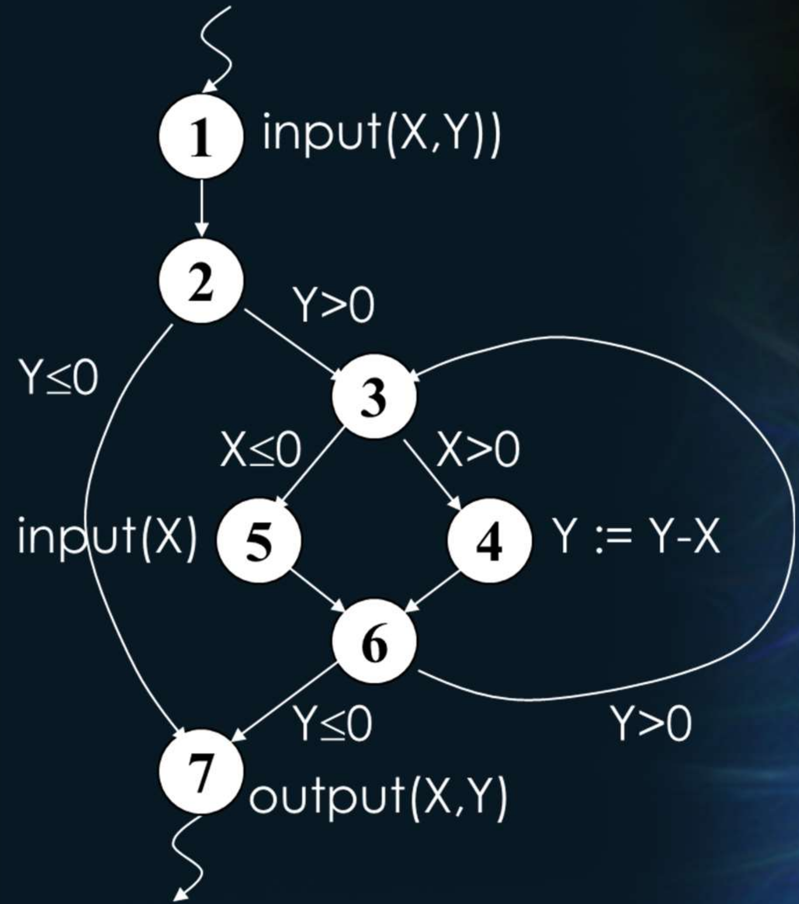
NOTE!

Identifying du-paths

c-use

du-pair	path(s)	du-path?
X: (1,4)	<1,2,3,4>	Yes
	<1,2,3,4,(6,3,4)*>	No
X: (1,7)	<1,2,7>	Yes
	<1,2,3,4,6,7>	Yes
	<1,2,3,4,6,(3,4,6)*,7>	No
X: (1,<3,4>)	<1,2,3,4>	Yes
	<1,2,3,4,(6,3,4)*>	No
X: (1,<3,5>)	<1,2,3,5>	Yes
X: (5,4)	<5,6,3,4>	Yes
	<5,6,3,4,(6,3,4)*>	No

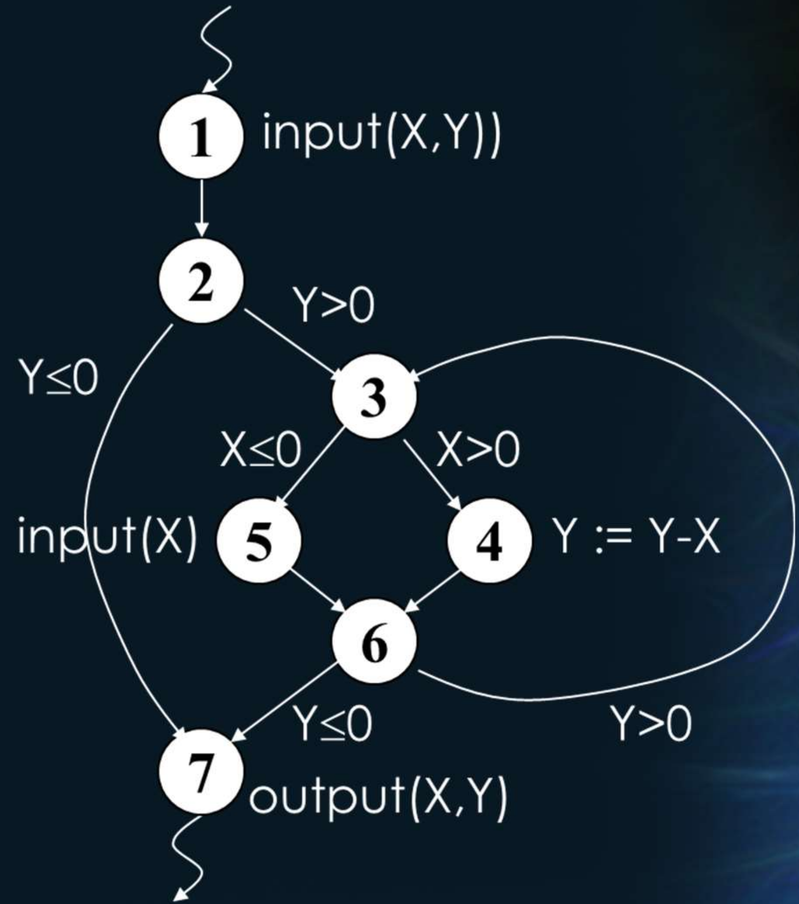
p-use



s1

Identifying du-paths

du-pair	path(s)	du-path?
X: (5,7)	<5,6,7> †	Yes
simple	<5,6,3,4,6,7>	Yes
	<5,6,3,4,6,(3,4,6)*,7>	No
X: (5,<3,4>)	<5,6,3,4>	Yes
	<5,6,3,4,(6,3,4)*>	No
X: (5,<3,5>)	<5,6,3,5>	Yes
Loop free	<5,6,3,4,6,3,5> †	No
	<5,6,3,4,6,(3,4,6)*,3,5> †	No
† infeasible		



幻灯片 68

s1 suda, 2022/10/5

Another Dataflow Test Coverage Criterion

- ***All-DU-Paths:*** for **every program variable v,** **every** **du-path** from **every definition** of v to **every c-use** and **every p-use** of v must be covered.

Exercise

- Identify du pairs, all c-uses and p-uses for **variable Y** in Exercise.
- Identify whether or not each def-clear path is feasible, and whether or not it is a du-path.

- Finally, receive the test suite to satisfy ***All-DU-Paths*** of this program.

Recall

- ***All-Defs:*** for every program variable v, at least one def-clear path from every definition of v to at least one c-use or one p-use of v must be covered.
- ***All-Uses:*** for every program variable v, at least one def-clear path from every definition of v to every c-use and every p-use of v must be covered.
- ***All-DU-Paths:*** for every program variable v, every du-path from every definition of v to every c-use and every p-use of v must be covered.

Summary of White-Box Coverage Relationships

The Commission Problem

A rifle salesperson sells rifle locks, stocks, and barrels. Locks cost $45, stocks cost $30, and barrels cost $25. The salesperson had to sell at least one complete rifle per month, and production limits were such that the most the salesperson could sell in a month was 70 locks, 80 stocks, and 90 barrels.

Suppose the commission is computed as follows: 10% on sales up to (and including) $1000, 15% on the next $800, and 20% on any sales in excess of $1800.

The commission program produces a monthly sales report that gave the total number of locks, stocks, and barrels sold, the salesperson's total dollar sales, and, finally, the commission.

佣金程序的问题描述

- 步枪销售商在亚利桑那州境内销售制造商制造的步枪锁、准星和枪管
- 枪锁卖45美元，准星卖30美元，枪管卖25美元
- 销售商每月至少要售出一枝完整的步枪，生产限额考虑到大多数销售商在一个月内可销售70个枪锁、80个准星和90个枪管
- 销售商在每访问一个镇子之后，给制造商发出电报，说明在那个镇子中售出的枪锁、准星和枪管数量

佣金程序的问题描述（续）

- 到了月末，销售商要发出一封很短的电报，通知－1个枪锁被售出，以便制造商知道当月的销售情况
- 销售商的佣金为：销售额不到（含）1000美元的部分，为10％，1000（不含）到1800（含）美元的部分，为15％，超过1800美元的部分为20％
- 佣金程序生成月份销售报告，汇总售出的枪锁、准星和枪管总数，销售商的总销售额，以及佣金

```
Program Commission (INPUT,OUTPUT)
    Dim locks,stocks,barrels As Integer
    Dim lockPrice,stockPrice,barrelPrice As Real
    Dim totalLocks,totalStocks,totalBarrels As Integer
    Dim lockSales,stockSales. barrelSales As Real
    Dim sales,commission As Real
    lockPrice = 45.0
    stockPrice = 30.0
    barrelPrice = 25.0
    totalLocks = 0
    totalStocks = 0
    totalBarrels = 0
```

```
Input(locks)
While NOT(locks = -1)     'loop condition uses-1 to indicate end of
    Input(stocks,barrels)
    totalLocks = totalLocks + locks
    totalStocks = totalStocks + stocks
    totalBarrels = totalBarrels + barrels
    Input(locks)
EndWhile
Output("Locks sold:",totalLocks)
Output("Stocks sold:",totalStocks)
Output("Barrels sold:",totalBarrels)
lockSales = lockPrice * totalLocks
stockSales = stockPrice * totalStocks
barrelSales = barrelPrice * totalBarrels
sales = lockSales + stockSales + barrelSales
Output("Total sales:",sales)
```

```
If(sales>1800.0)
    Then
        commission = 0.10 * 1000.0
        commission = commission + 0.15 * 800.0
    commission = commission + 0.20 * (sales - 1800.0)
Else If(sales>1000.0)
    Then
            commission = 0.10 * 1000.0
            commission = commission + 0.15 * (sales - 1000.0)
        Else commission = 0.10 * sales
          EndIf
 EndIf
 Output("Commission is $",commission)
 End Commission
```

Homework

- Draw control flow graph.
- For these variable: LockPrice, TotalStocks, Barrels, LockSales, Sales, Commission
  - List du-pairs and def-clear paths
  - Design test cases to satisfy all du-path coverage.